Adding Optimization to Your Applications
Quickly and Reliably

1. A Guide to Model-Based Optimization

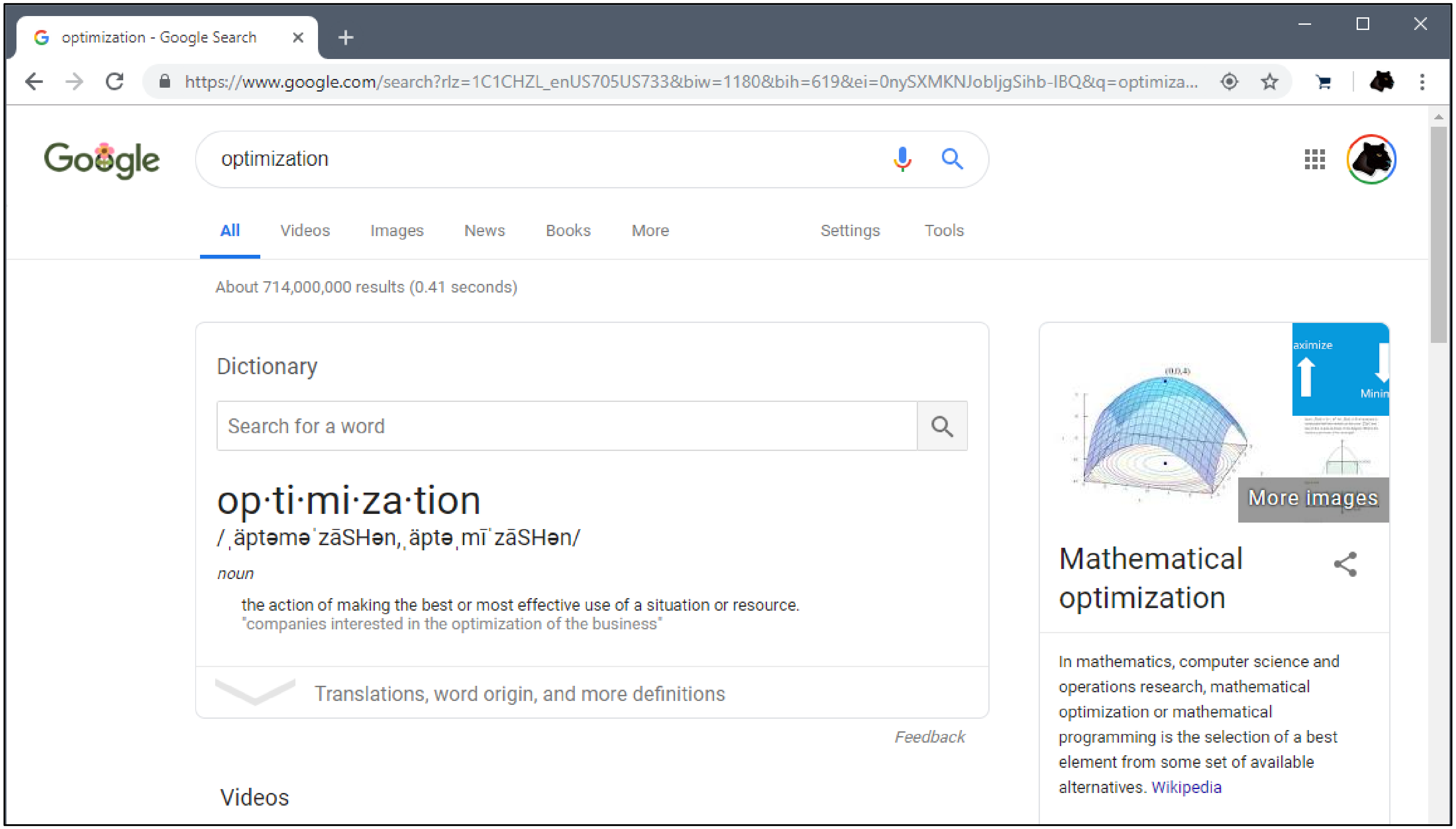
2. From Prototyping to Integration with AMPL

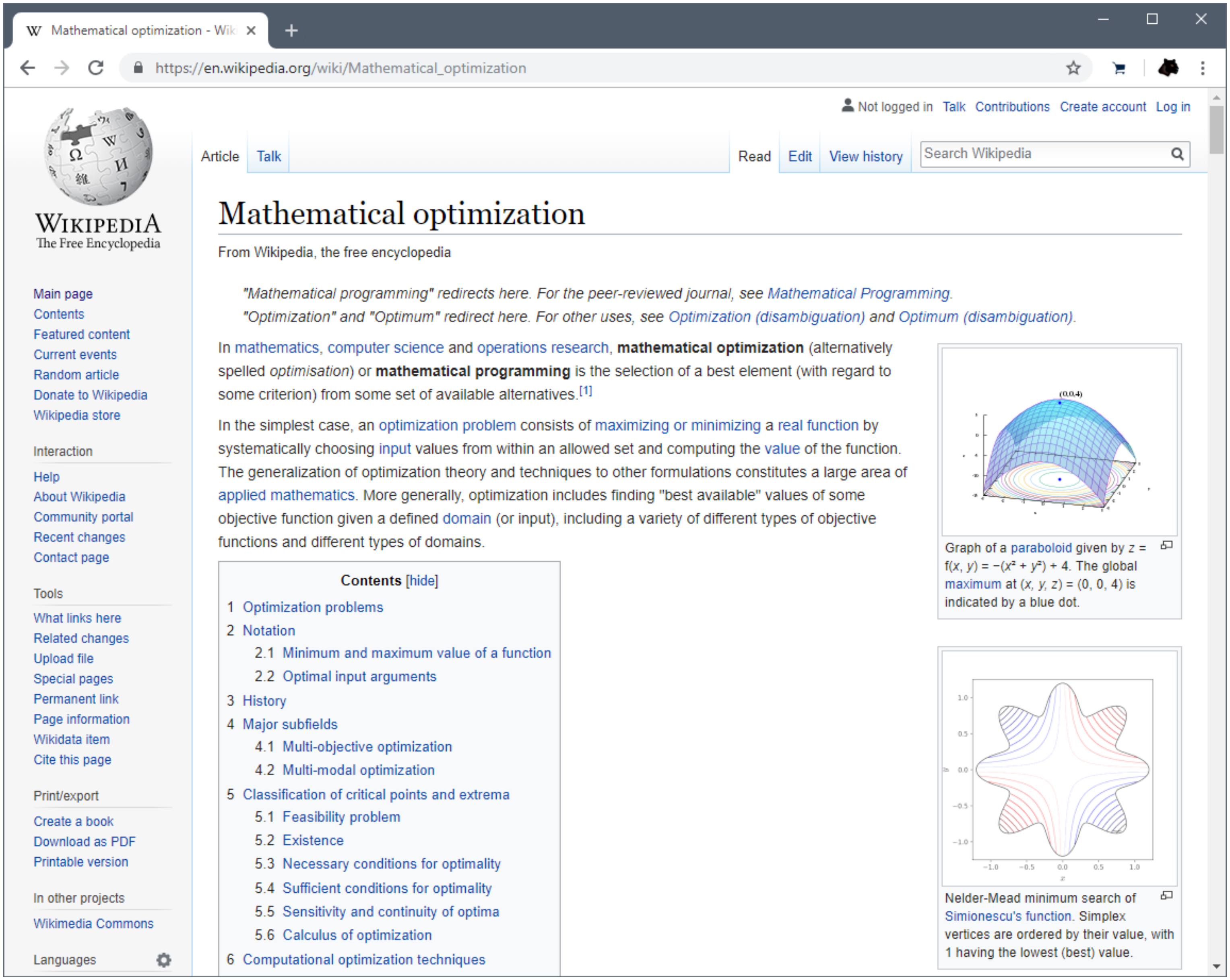
AMPL Optimization Inc.

www.ampl.com — +1 773-336-AMPL

INFORMS Business Analytics Conference

Technology Workshop, 14 April 2019


optimization - Google Search
https://www.google.com/search?rlz=1C1CHZL_enUS705US733&biw=1180&bih=619&ei=0nySXMKNJobljgSihb-IBQ&q=optimiza...
optimization
All
Videos
Images
News
Books
More
Settings
Tools
About 714,000,000 results (0.41 seconds)
Dictionary
Search for a word
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
Videos
More images
Mathematical optimization
In mathematics, computer science and operations research, mathematical optimization or mathematical programming is the selection of a best element from some set of available alternatives. Wikipedia


Mathematical optimization - Wiki

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
Wikimedia Commons

Languages

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** (alternatively spelled *optimisation*) or **mathematical programming** 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 vertices are ordered by their value, with 1 having the lowest (best) value.

Contents [hide]

Mathematical Optimization

In general terms,

- Given an objective function of some decision variables
- Choose values of the variables to
  make the objective 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 *decision problems*
- A practical approach to making decisions

Outline

1. Model-based optimization

- Comparison of *method-based* and *model-based* approaches
- Modeling languages for optimization
- Algebraic modeling languages: AMPL
- Off-the-shelf solvers for common model types

2. From prototyping to integration

- Building models: *AMPL's interactive environment*
- Developing optimization-based procedures: *AMPL scripts*
- Integrating into decision-making systems: *AMPL APIs*
  - Integrating with Python applications: *pyMPL*
  - Building a decision-making tool for deployment: *QuanDec*

Example: 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

Balanced Assignment

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

   GMin = GMin ∪ {p}
```

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 solvers
- Broad problem classes handled efficiently
- Special cases recognized and exploited to advantage
  - zero-one variables like x_{ij}

Balanced Assignment

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

$x_{ij} \in \{0,1\}$ $\quad = 1$ if person i is assigned to group j
$\quad\quad\quad\quad\quad\quad = 0$ 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

$g^{\min} \le \sum_{i \in P} x_{ij} \le g^{\max}$, 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, MIPCL, SCIP
- 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, . . .

Model-Based vs. Method-Based

Where is the work?

- *Method-based:* Programming an implementation of the method
- *Model-based:* Constructing a formulation of the model

Which should you prefer?

- For simple problems, any approach can seem pretty easy
- *But real optimization problems are seldom simple . . .*

Complications in Balanced Assignment

"Total Sameness" is hard to relate to the goal of diversity

- *Minimize "total variation" instead*
  - Sum over all types: most minus least assigned to any group

No employee should feel "isolated" within their group

- No group should have exactly one woman
- Every person should 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 solution approach

- Total variation is less suitable to a greedy algorithm
- 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

Balanced Assignment

Model-Based *(cont'd)*

Replace the objective

Formulate additional constraints

Send back to the solver

Model-Based *(cont'd)*

To write new objective, 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

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

Balanced Assignment

Model-Based *(cont'd)*

To express client requirement for women in a group, let

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

Add constraints

$\sum_{i \in Q} x_{ij} = 0$ or $\sum_{i \in Q} x_{ij} \geq 2$, 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 variables to assignment variables

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

Balanced Assignment

Model-Based *(cont'd)*

To express client requirement for women in a group, let

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

Define logic variables

$z_j \in \{0,1\} = 1$ if any women assigned to group j

$= 0$ otherwise, for all $j = 1, \ldots, G$

Linearize constraints relating logic variables to assignment variables

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

Model-Based *(cont'd)*

To express client requirements for group-mates, let

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

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

Add constraints

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

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

Balanced Assignment

Model-Based *(cont'd)*

To express client requirements for group-mates, let

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

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

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

Add constraints relating logic variables to assignment variables

$w_{l_1 l_2 j} = 0 \Rightarrow \sum_{i \in R_{l_1 l_2}} x_{ij} = 0,$

$w_{l_1 l_2 j} = 1 \Rightarrow \sum_{i \in R_{l_1 l_2}} x_{ij} + \sum_{l \in A_{l_2}} \sum_{i \in R_{l_1 l}} x_{ij} \geq 2\,,$

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

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

$A_l \subseteq T_{\mathrm{rank}}$, set of 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 $l_1 \in T_{\mathrm{loc}}, l_2 \in T_{\mathrm{rank}}, j = 1,\ldots,G$

Linearize constraints relating logic variables to assignment variables

$w_{l_1 l_2 j} \leq \sum_{i \in R_{l_1 l_2}} x_{ij} \leq |R_{l_1 l_2}|\, w_{l_1 l_2 j}$,

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

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

Method-Based Remains Popular for . . .

Heuristic approaches

- Simple heuristics
  - Greedy algorithms, local improvement methods
- Metaheuristics
  - Evolutionary methods, simulated annealing, tabu search, GRASP, . . .

Situations hard to formulate mathematically

- Difficult combinatorial constraints
- Black-box objectives and constraints

Large-scale, intensive applications

- Routing fleets of delivery trucks
- Finding shortest routes in mapping apps

. . . and it appeals to programmers

Model-Based Has Become Common for . . .

Diverse application areas *(active AMPL users)*

- Energy and Utilities
  - power networks, gas pipelines, hydroelectric power, water distribution
- Industry
  - mining, steel, chemicals, oil refining, forestry and paper
  - cars & trucks, paper products, processed foods
- Transportation
  - airlines, trucking
- Services
  - supply chain, hospitals & medicine, construction management
- Communications
  - telecommunications, social media, cloud computing, distribution
- Finance
  - software tools, investment management, commodity management
- Advanced Technologies
  - artificial intelligence, distributed computing, biotechnology

Model-Based Has Become Common for . . .

Diverse application areas

Diverse fields

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

Model-Based Has Become Common for . . .

Diverse industries

Diverse fields

Diverse kinds of users

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

These have in common . . .

- ❖ Users inclined toward modeling; focus is
  - ∗ more on *what* should be solved
  - ∗ less on *how* it should be solved
- ❖ Good algebraic formulations for off-the-shelf solvers

Trends Favor Model-Based Optimization

Model-based approaches have spread

- Model-based metaheuristics ("Matheuristics")
- Solvers for SAT, planning, constraint programing

Off-the-shelf optimization solvers have kept improving

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

Optimization models have become easier to embed within broader methods

- Model-based evolution of solver APIs
- APIs for optimization modeling systems

Software for Model-Based Optimization

Background

- The modeling lifecycle
- Matrix generators
- Modeling languages

Algebraic modeling languages

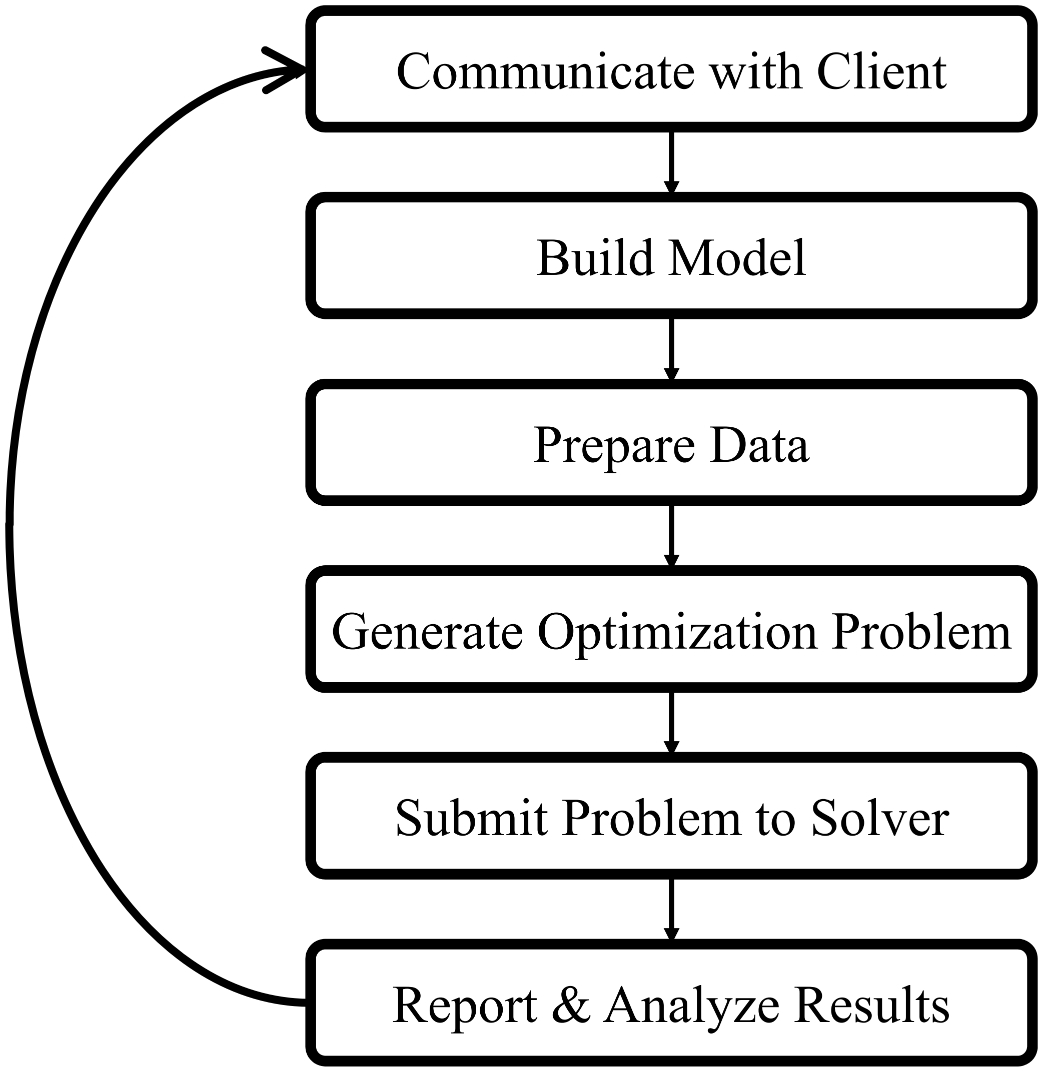
- Solver-independent vs. solver-specific
- Declarative vs. executable

Examples

- A simple transportation model, in AMPL and gurobipy
- The balanced assignment model revisited, in AMPL

Solvers

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

Algebraic formulation

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

Advantages

- Familiar
- Powerful
- Proven

Algebraic Modeling Languages

Design approaches

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

Marketing approaches

- Solver-independent vs. solver-specific
- Licensed vs. open-source

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 languages are not designed for describing models
  - Additional documentation may be needed to explain constraints
  - Special methods may be required for efficiency
- Modeling and programming bugs are hard to separate

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 an object-oriented programming framework

Advantages

- Streamlines model development
- Promotes validation and maintenance of models
- Can provide APIs for many popular programming languages

Declarative

Many enhancements and extensions

- Interactive development environments
- Generalized constraint forms
- Variety of data sources
  - spreadsheets, relational databases
- Programming features
  - loops, tests, assignments
- Extensions for deployment
  - APIs for embedding models in applications
  - Tools for building applications around models

AMPL

Features

- Algebraic modeling language
- Built specially for optimization
- Designed to support many solvers

Design goals

- Powerful, general expressions
- Natural, easy-to-learn modeling principles
- Efficient processing that scales well with problem size

Example: A Simple Transportation Model

Motivation

- Ship commodities through a distribution network
  - Shipment origins/destinations are *nodes* of the network
  - Shipment possibilities are *arcs* connecting the nodes
- Each commodity has supplies and demands at various nodes

Optimization model

- Decision variables
  - amount of each commodity to ship over each arc
- Objective
  - minimize total cost of shipments
- Constraints
  - balance commodity in vs. commodity out at each node
  - satisfy shipping capacity on each arc

Algebraic Formulation

Given

H set of commodities

N set of network nodes

$A \subseteq N \times N$ set of arcs connecting nodes

and

u_{ij} capacity of arc from i to j, for each $(i, j) \in A$

s_{hj} supply/demand of commodity h at node i, for each $h \in H$, $j \in N$
> 0 implies supply, < 0 implies demand

c_{hij} cost per unit to ship commodity h on arc (i, j),
for each $h \in H$, $(i, j) \in A$

Algebraic Formulation *(cont'd)*

Determine

X_{hij} amount of commodity h to be shipped on arc (i, j),
for each $h \in H$, $(i, j) \in A$

to minimize

$\sum_{h \in H} \sum_{(i,j) \in A} c_{hij} X_{hij}$

total cost of shipments

subject to

$\sum_{h \in H} X_{hij} \leq u_{ij}$, for all $(i, j) \in A$

total shipments on each arc must not exceed capacity

$\sum_{(i,j) \in A} X_{hij} + s_{hj} = \sum_{(j,i) \in A} X_{hji}$, for all $h \in H$, $j \in N$

shipments in plus supply/demand must equal shipments out

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 })
```

AMPL

- Define symbolic model sets and parameters

```
set COMMODITIES;
set NODES;

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

- Provide data later in a separate file

```
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];
```

$$\sum_{(i,j)\in A} X_{hij} + s_{hj} = \sum_{(j,i)\in A} X_{hji}, \quad \text{for all } h \in H, j \in N$$

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]))
```

```
Solved in 0 iterations and 0.00 seconds
Optimal objective  5.500000000e+03

Optimal flows for Pencils:
Detroit -> Boston: 50
Denver -> New York: 50
Denver -> Seattle: 10

Optimal flows for Pens: ...
```

Comparison

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
;
```

Comparison

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

Comparison

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

Balanced Assignment Revisited

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

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

set of all types of people in category k

Balanced Assignment Revisited *in AMPL*

Sets, parameters

```
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


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

              # all types found in each category
```

Balanced Assignment

Determine

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

$y_{kl}^{\min}$ 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 $k \in C$, $l \in T_k$

Where

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

Balanced Assignment *in AMPL*

Variables, defining constraints

```
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

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
```

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

Balanced Assignment

Minimize

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

sum of inter-group variation over all types in all categories

Subject to

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

each person must be assigned to one group

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

each group must be assigned an acceptable number of people

Balanced Assignment *in AMPL*

Objective, assignment constraints

```
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
```

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

Balanced Assignment

Define also

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

Determine

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

Subject to

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

each group must have either
no women ($z_j = 0$) *or* ≥ 2 *women* ($z_j = 1$)

Balanced Assignment *in AMPL*

Supplemental constraints

```
set WOMEN = {i in PEOPLE: type[i,'m/f'] = 'F'};

var WomenInGroup {j in 1..numberGrps} binary;

subj to Min2WomenInGroupLO {j in 1..numberGrps}:
   2 * WomenInGroup[j] <= sum {i in WOMEN} Assign[i,j];

subj to Min2WomenInGroupUP {j in 1..numberGrps}:
   sum {i in WOMEN} Assign[i,j] <= card(WOMEN) * WomenInGroup[j];
```

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

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, 12 groups, 16-19 people/group

```
set CATEG := dept loc 'm/f' title ;

param type:
     dept       loc     'm/f'   title   :=

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

2568 variables:
        2532 binary variables
        36 linear variables
678 constraints, all linear; 26328 nonzeros
        210 equality constraints
        456 inequality constraints
        12 range constraints
1 linear objective; 36 nonzeros.

CPLEX 12.9.0.0: optimal integer solution; objective 16
23690 MIP simplex iterations
159 branch-and-bound nodes
```

7.4 sec

Balanced Assignment

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;

2568 variables:
        2532 binary variables
        36 linear variables
678 constraints, all linear; 26328 nonzeros
        210 equality constraints
        456 inequality constraints
        12 range constraints
1 linear objective; 36 nonzeros.

Gurobi 8.1.0: optimal solution; objective 16
521639 simplex iterations
804 branch-and-cut nodes
```

103.2 sec

Balanced Assignment *(logical)*

Define also

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

Determine

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

Where

$$z_j = 0 \;\Rightarrow\; \textstyle\sum_{i \in Q} x_{ij} = 0,$$

$$z_j = 1 \;\Rightarrow\; \textstyle\sum_{i \in Q} x_{ij} \ge 2\,, \text{ for each } j = 1, \ldots, G$$

Balanced Assignment *in AMPL*

Supplemental logical constraints

```
set WOMEN = {i in PEOPLE: type[i,'m/f'] = 'F'};

var WomenInGroup {j in 1..numberGrps} binary;

subj to Min2WomenInGroup {j in 1..numberGrps}:

   WomenInGroup[j] = 0  ==>  sum {i in WOMEN} Assign[i,j] = 0
                            else sum {i in WOMEN} Assign[i,j] >= 2;
```

$$z_j = 0 \Rightarrow \sum_{i \in Q} x_{ij} = 0,$$

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

Balanced Assignment *in AMPL*

Send to "linear" solver

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

ampl: option solver gurobi;
ampl: solve

2568 variables:
        2184 binary variables
        348 nonlinear variables
        36 linear variables
654 algebraic constraints, all linear; 25632 nonzeros
        210 equality constraints
        432 inequality constraints
        12 range constraints
12 logical constraints
1 linear objective; 29 nonzeros.

Gurobi 8.1.0: optimal solution; objective 16
409935 simplex iterations
798 branch-and-cut nodes
```

68.6 sec

Balanced Assignment *in AMPL (refined)*

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: solve

Presolve eliminates 72 constraints.
...

Gurobi 8.1.0: optimal solution; objective 16
1022 simplex iterations
1 branch-and-cut nodes
```

0.14 sec

Modeling Language Solution

Result display script

```
param typelen {k in CATEG} = max {l in TYPES[k]} length(l) + 2;

for {j in 1..numberGrps} {
   printf "GROUP %i\n\n", j;
   for {i in PEOPLE: Assign[i,j] = 1} {
      printf "%-6s", i;
      printf {k in CATEG}: "%-*s", typelen[k], type[i,k];
      printf "\n";
   }
   printf "\n";
}

for {k in CATEG}
   display {j in 1..numberGrps, l in TYPES[k]}
      sum {i in PEOPLE: type[i,k] = l} Assign[i,j];

display {j in 1..numberGrps} sum {i in PEOPLE} Assign[i,j];
```

Solvers for Model-Based Optimization

Off-the-shelf solvers for broad problem classes

Three widely used types

- “Linear”
- “Nonlinear”
- “Global”

"Linear" Solvers

Require objective and constraint coefficients

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

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

“Nonlinear” Solvers

Require function and derivative evaluations

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

Require expression graphs (or equivalent)

Nonlinear + global optimality

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

BARON

- Dominant commercial global solver

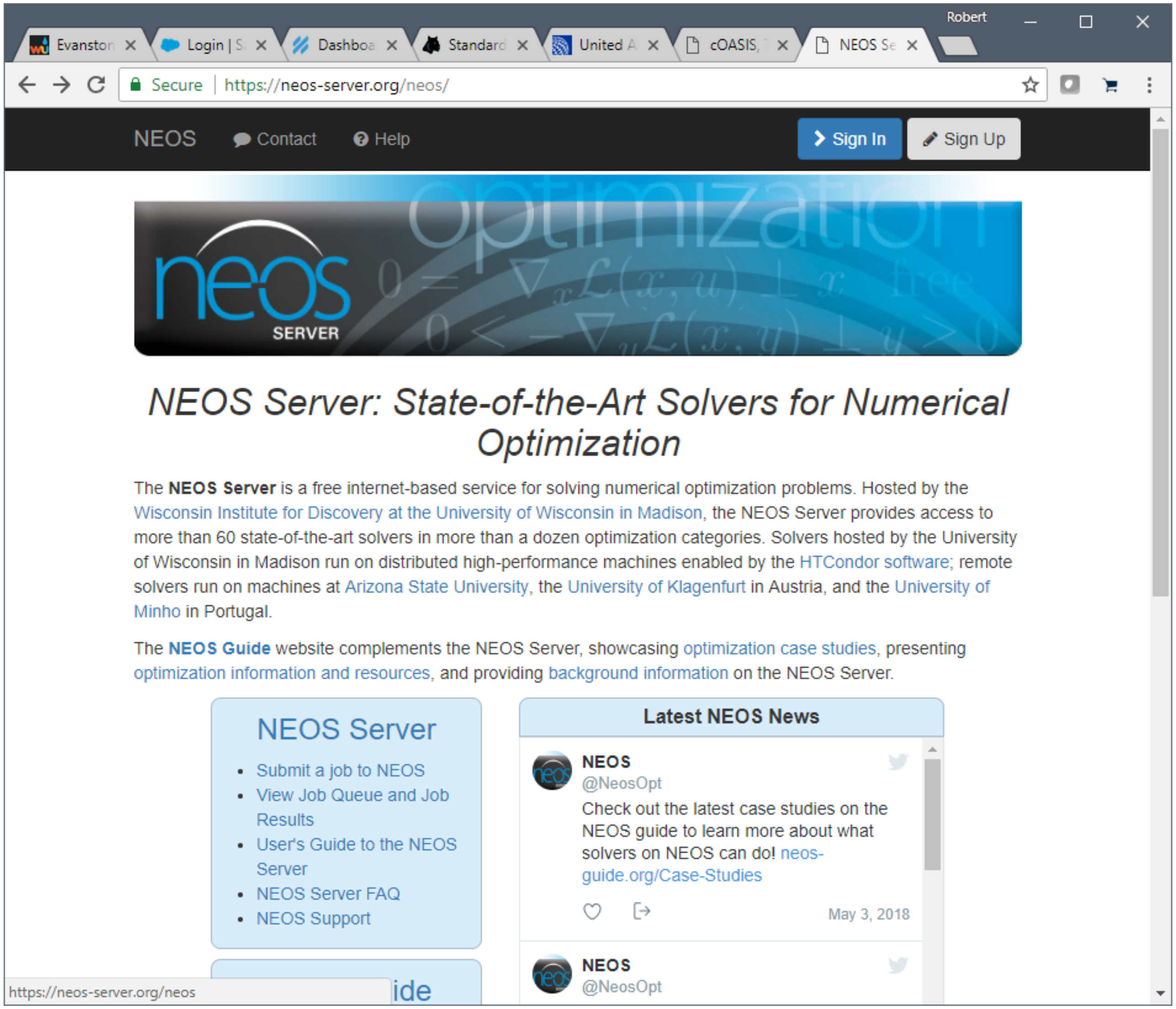
Couenne

- Highly regarded noncommercial global solver

LGO

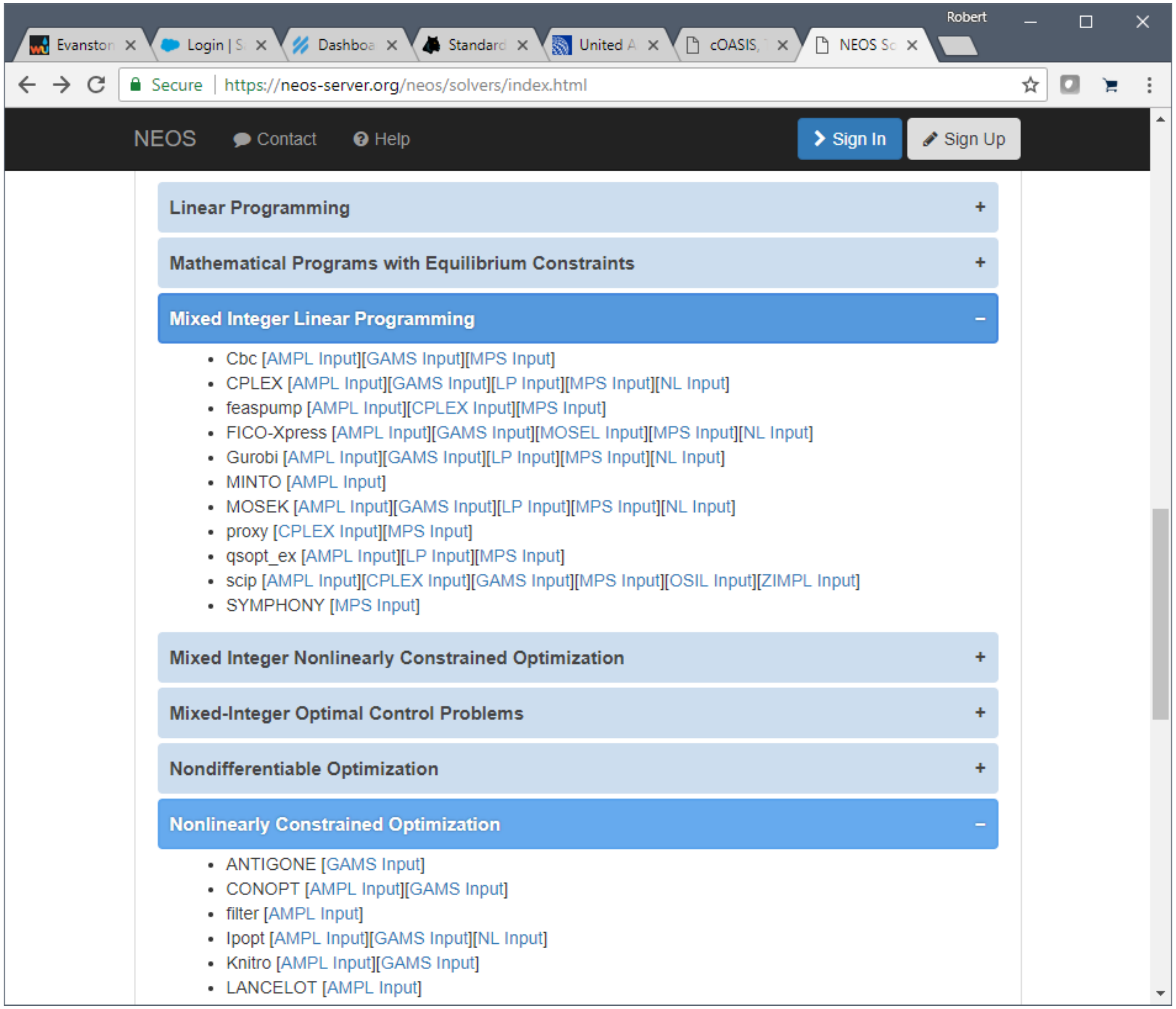
- High-quality solutions, may be global
- Objective and constraint functions may be nonsmooth

Curious? Try Them Out on NEOS!



NEOS Server

Solver & Language Listing



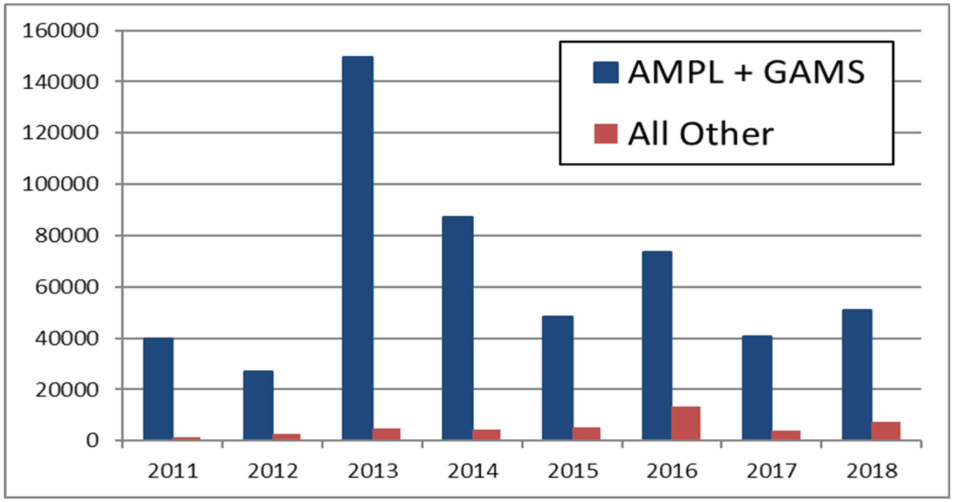
About the NEOS Server

Solvers

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

Inputs

- Modeling languages: AMPL, GAMS, . . .
- Lower-level formats: MPS, LP, . . .



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)