Adding Optimization to Your Applications, from Prototyping to Deployment

How AMPL is Making It Faster and Easier

1. A Guide to Model-Based Optimization

2. From Prototyping to Integration with AMPL

Robert Fourer

`4er@ampl.com`

AMPL Optimization Inc.
www.ampl.com — +1 773-336-AMPL

INFORMS Annual Meeting
Technology Workshop, 15 October 2022

Adding Optimization to Your Applications, from Prototyping to Deployment: How AMPL is Making It Faster and Easier

Optimization is the most widely adopted technology of Operations Research and Analytics, but also the most challenging to implement:

- How can you *prototype* an optimization application fast enough to get results before the problem owner loses interest?
- How can you *develop* optimization-based procedures to get results you can use, within your time and resource requirements?
- How can you *integrate* optimization into your enterprise's decision-making systems?
- How can you *deploy* optimization in modern cloud and container environments?

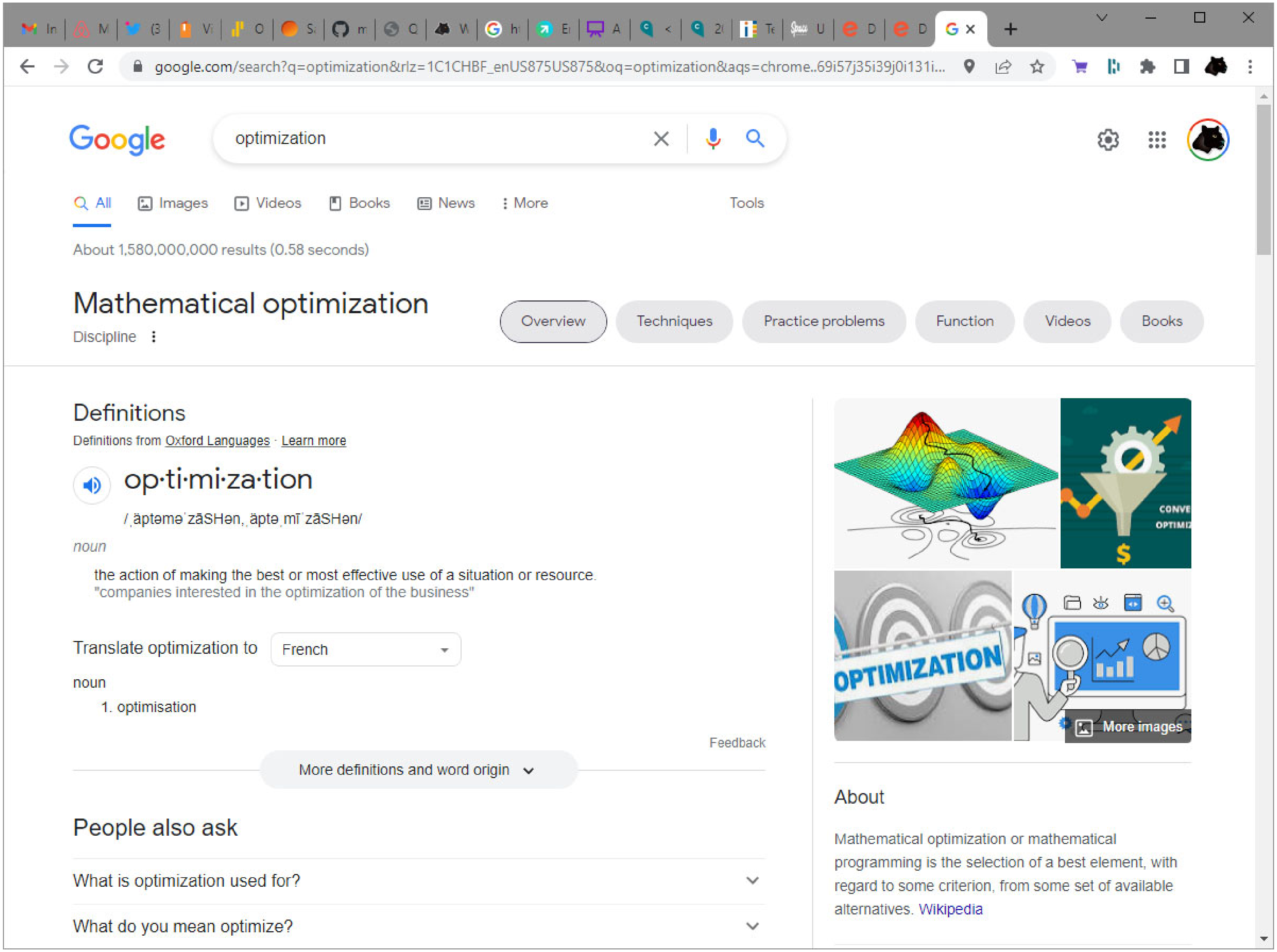
In this presentation, we show how AMPL gets you going without elaborate training, extra programmers, or premature commitments. We specially highlight ***new AMPL features*** that make optimization modeling faster, easier, and more effective than ever.

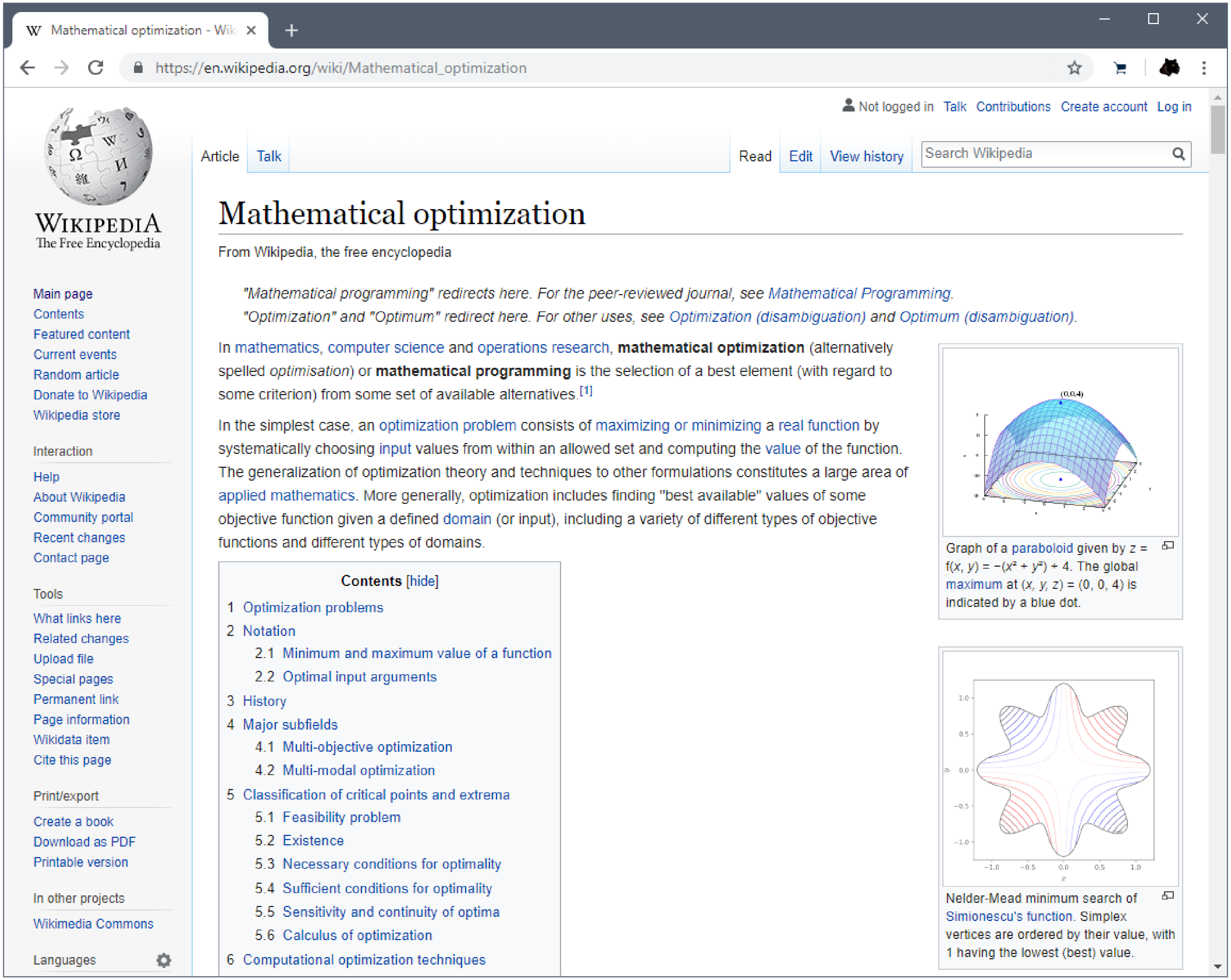
We start by introducing *model-based optimization,* the key approach to streamlining the optimization modeling cycle and building successful applications today. Then we demonstrate how AMPL's design of a language and system for model-based optimization is able to offer exceptional power of expression and speed of execution while maintaining ease of use.

The presentation continues by taking a single example through successive stages of the optimization modeling lifecycle, highlighting recent enhancements:

- *Prototyping* in an interactive command environment, with a new solver interface that accepts more natural modeling language expressions
- *Development* of optimization procedures via AMPL's built-in scripting language and enhanced spreadsheet/database interfaces
- *Integration* through APIs to widely used programming languages, including Python with new notebook support
- *Deployment* using new, flexible installation and licensing support

Our example is simple enough for participants to follow its development through the course of this short workshop, yet rich enough to serve as a foundation for appreciating model-based optimization in practice. Several case studies of AMPL applications round out the presentation by showing how model-based optimization has been successful in varied areas of analytics practice.


google.com/search?q=optimization&rlz=1C1CHBF_enUS875US875&oq=optimization&aqs=chrome..69i57j35i39j0i131i...
Google
optimization
All
Images
Videos
Books
News
More
Tools
About 1,580,000,000 results (0.58 seconds)
Mathematical optimization
Discipline
Overview
Techniques
Practice problems
Function
Videos
Books
Definitions
Definitions from Oxford Languages · Learn more
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"
Translate optimization to
French
noun
1. optimisation
Feedback
More definitions and word origin
People also ask
What is optimization used for?
What do you mean optimize?
OPTIMIZATION
More images
About
Mathematical optimization or mathematical programming is the selection of a best element, with regard to some criterion, from some set of available alternatives. Wikipedia



Mathematical optimization - Wik

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

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

Not logged in Talk Contributions Create account Log in

Article Talk Read Edit View history Search Wikipedia

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 concept,

- 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 constraints on the values of the variables

In practice,

- A paradigm for a very broad variety of *decision problems*
- A valuable approach to making decisions

Optimization in OR & Analytics

Given a recurring need to make many interrelated decisions

- Purchases, production and shipment amounts, assignments, . . .

Consistently make highly desirable choices

By applying ideas from mathematical optimization

- Ways of describing problems *(models)*
- Ways of solving problems *(algorithms)*

Optimization in Practice

Large numbers of decision variables

- Thousands to millions

An objective function

- To be minimized or maximized

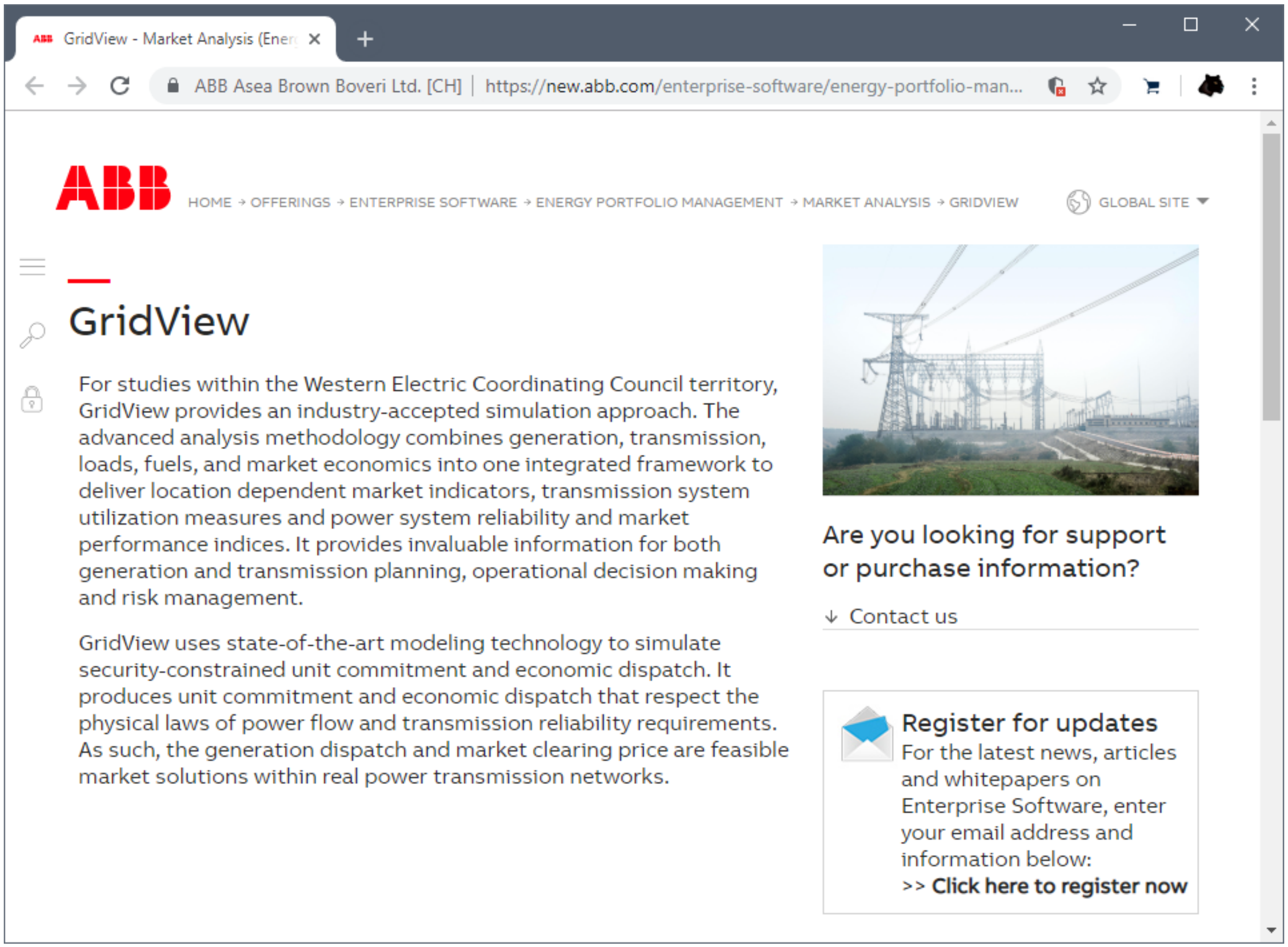
Various constraint types

- 10-20 distinct types
- Thousands to millions of each type
- Few variables involved in each constraint

Solved many times with different data

- Using large-scale, general-purpose software
- Built on iterative optimization algorithms

Case: ABB
Managing Power Grids



GridView - Market Analysis (Ener

ABB Asea Brown Boveri Ltd. [CH] | https://new.abb.com/enterprise-software/energy-portfolio-man...

HOME → OFFERINGS → ENTERPRISE SOFTWARE → ENERGY PORTFOLIO MANAGEMENT → MARKET ANALYSIS → GRIDVIEW

GLOBAL SITE

GridView

For studies within the Western Electric Coordinating Council territory, GridView provides an industry-accepted simulation approach. The advanced analysis methodology combines generation, transmission, loads, fuels, and market economics into one integrated framework to deliver location dependent market indicators, transmission system utilization measures and power system reliability and market performance indices. It provides invaluable information for both generation and transmission planning, operational decision making and risk management.

GridView uses state-of-the-art modeling technology to simulate security-constrained unit commitment and economic dispatch. It produces unit commitment and economic dispatch that respect the physical laws of power flow and transmission reliability requirements. As such, the generation dispatch and market clearing price are feasible market solutions within real power transmission networks.

Are you looking for support or purchase information?

↓ Contact us

Register for updates

For the latest news, articles and whitepapers on Enterprise Software, enter your email address and information below:

>> **Click here to register now**

Case: ABB
Power Grid Management

GridView

For studies within the Western Electric Coordinating Council territory, GridView provides an industry-accepted simulation approach. The advanced analysis methodology combines generation, transmission, loads, fuels, and market economics into one integrated framework to deliver location dependent market indicators, transmission system utilization measures and power system reliability and market performance indices. It provides invaluable information for both generation and transmission planning, operational decision making and risk management.

GridView uses state-of-the-art modeling technology to simulate security-constrained unit commitment and economic dispatch. It produces unit commitment and economic dispatch that respect the physical laws of power flow and transmission reliability requirements. As such, the generation dispatch and market clearing price are feasible market solutions within real power transmission networks.

Power Grid

Application

Setting

- Power grid operators provide electrical service
- Two kinds of decisions
  - *Unit commitment:* When to turn power plants on and off
  - *Network flow:* How to transmit power over the grid to meet demand

Goal

- Envision optimal decisions to support planning
  - Transmission network expansion
  - Plant addition and retirement
  - Integration of renewable energy sources

Power Grid

Evaluation

Approaches considered

- C++ for entire GridView system
- Modeling language for optimization, C++ for user interfaces

Choice of AMPL

- *Ease of modeling*
  - ABB can formulate complex and powerful models
  - Customers can understand the AMPL formulations
  - Customers can specialize models to their particular situations
- *Ease of embedding*
  - Optimization can be built into the GridView product using AMPL's C++ API

Power Grid

Formulation *(data)*

Production data

- Power generation units
  - Location
  - Fuel, design, age, capacity
  - Ramp-up and ramp-down times
- Renewable energy sources

Transmission network data

- Nodes: units, sources, substations, customers
  - Supply at plants and other sources
  - Demand at customers
- Arcs: power lines
  - Transmission capacities

Cost data

Power Grid

Formulation *(variables)*

Decision variables

- For each unit, in each time period
  - On or off (discrete)
  - Level of output (continuous)
- For each *critical path* through the grid, in each time period
  - Capacity

Power Grid

Formulation *(model)*

Objectives

- For short-term operation management
  - Minimize total operating costs
- For long-term investment planning
  - Minimize total operating and investment costs

Constraints

- Balance of supply and demand
- Capacity restriction on power lines
- Ramp-up and ramp-down times
- Contingencies for generation and transmission

Implementation

Development

- Prototype at University of Tennessee, Knoxville
- Full AMPL implementation by three analysts at ABB

Optimization

- Mixed-integer linear solver
- Millions of variables
- Tens of thousands of integer variables
- 10 minutes to solve

Deployment

- 30+ customer companies
- Hundreds of customer-side users

Outline

1. Model-based optimization

- Comparison of *method-based* and *model-based* approaches
- Approaches to model-based optimization
- Approaches to algebraic modeling languages
- *AMPL*
  - ***formulate constraints like you think about them***
  - ***read & write common spreadsheet table formats***
- Ready-to-run solvers

2. From prototyping to integration

3. Case studies

Example: Balanced Assignment

Motivation

- meeting of employees from around the world

Given

- several employee categories
  (title, location, department, gender)
- 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 "varied" 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

   GMin = GMin ∪ {p}
```

Balanced Assignment

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?

Balanced Assignment

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 a ready-to-run solver

- Many excellent alternatives are available
- Broad problem classes are handled efficiently
- Special cases are 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

$\quad = 0$ 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

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

each person must be assigned to one group

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

each group must be assigned an acceptable number of people

Balanced Assignment

Model-Based Solution

Optimize with a ready-to-run 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
  - Global nonlinear: BARON, Lindo Global, Octeract
  - Mixed-integer local nonlinear: Knitro
  - Continuous nonlinear (might come out integer): MINOS, Ipopt, . . .

Model-Based vs. Method-Based

Which gets better results?

- *Method-based:*
  Speedy problem-specific heuristics, but no optimality guarantee
- *Model-based:*
  Provably (near-)optimal solutions, but no speed guarantee

But this is not the main issue . . .

Model-Based vs. Method-Based

Where is the effort?

- *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

Client has trouble with "Total Sameness"

- Hard to relate to goal of having varied groups
- *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 objective is less suitable to a simple algorithm
- Isolation 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)*

Update the model

- Replace the objective with "total variation"
- Add "isolation" constraints

Re-run the solver

- Total variation is actually easier

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

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

Define logic variables

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

Add constraints relating logic variables to assignment variables

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

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

Model-Based *(cont'd)*

To express client requirement for women in a group, let

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

Define logic variables

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

Linearize constraints relating
logic variables to assignment variables

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

Balanced Assignment

Model-Based *(cont'd)*

To express client requirements for group-mates, let

$LR_{lr} = \{i \in P: t_{i,\mathrm{loc}} = l,\ t_{i,\mathrm{rank}} = r\}$, for all $l \in T_{\mathrm{loc}}, r \in T_{\mathrm{rank}}$

$A_r \subseteq T_{\mathrm{rank}}$, set of ranks adjacent to rank r, for all $r \in T_{\mathrm{rank}}$

Add constraints

$\sum_{i \in LR_{lr}} x_{ij} = 0$ or $\sum_{i \in LR_{lr}} x_{ij} + \sum_{a \in A_r} \sum_{i \in LR_{la}} x_{ij} \geq 2$,

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

Model-Based *(cont'd)*

To express client requirements for group-mates, let

$LR_{lr} = \{i \in P\colon t_{i,\text{loc}} = l,\ t_{i,\text{rank}} = r\}$, for all $l \in T_{\text{loc}}, r \in T_{\text{rank}}$

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

Define logic variables

$w_{lrj} \in \{0,1\}$ $= 1$ if group j has anyone from location l of rank r
$= 0$ otherwise, for all $l \in T_{\text{loc}}, r \in T_{\text{rank}},\ j = 1,\ldots,G$

Add constraints relating logic variables to assignment variables

$$w_{lrj} = 0 \Rightarrow \textstyle\sum_{i \in LR_{lr}} x_{ij} = 0,$$

$$w_{lrj} = 1 \Rightarrow \textstyle\sum_{i \in LR_{lr}} x_{ij} + \sum_{a \in A_r} \sum_{i \in LR_{la}} x_{ij} \geq 2\,,$$

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

Model-Based *(cont'd)*

To express client requirements for group-mates, let

$LR_{lr} = \{i \in P: t_{i,\mathrm{loc}} = l,\ t_{i,\mathrm{rank}} = r\}$, for all $l \in T_{\mathrm{loc}}, r \in T_{\mathrm{rank}}$

$A_r \subseteq T_{\mathrm{rank}}$, set of ranks adjacent to rank r, for all $r \in T_{\mathrm{rank}}$

Define logic variables

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

Linearize constraints relating logic variables to assignment variables

$w_{lrj} \le \sum_{i \in LR_{lr}} x_{ij} \le |LR_{lr}|\, w_{lrj},$

$\sum_{i \in LR_{lr}} x_{ij} + \sum_{a \in A_r} \sum_{i \in LR_{la}} x_{ij} \ge 2 w_{lrj},$

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

Method-Based Use Cases . . .

Your problem is hard to describe mathematically

You favor a problem-specific heuristic method

You have a very large programming budget

. . . or you want to practice programming algorithms

Model-Based Use Cases . . .

Diverse industries

- Manufacturing, distribution, supply-chain management
- Air and rail operations, trucking, delivery services
- Medicine, medical services
- Refining, electric power flow, gas pipelines, hydropower
- Finance, e-commerce, . . .

Model-Based Use Cases . . .

Diverse industries

Diverse fields

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

Model-Based Use Cases . . .

Diverse industries

Diverse fields

Diverse kinds of users

- ❖ OR and analytics experts
- ❖ Anyone who took an "optimization" class
- ❖ Anyone else with a technical background

These have in common . . .

- ❖ Analysts inclined toward modeling; focus is
  - ∗ more on *what* should be solved
  - ∗ less on *how* it should be solved
- ❖ Good algebraic formulations for ready-to-run solvers
- ❖ Emphasis on fast prototyping *and* continued revision

Trends Favor Model-Based Optimization

Model-based approaches have spread

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

Ready-to-run 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

- Solver APIs that are model model-based
- APIs for optimization modeling systems

Approaches to Model-Based Optimization

Translate between two forms of the problem

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

Programming language approach

- Write a *computer program* to generate the solver's form

Modeling language approach

- Write the *model formulation*
  in a form that a computer can read and translate

Programming Language Approach

Write a program to generate the solver's form

- Read data and compute objective & constraint coefficients
- Send the solver the data structures it needs
- Receive solution data structure 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

Modeling Language Approach

Use a computer language to describe the modeler's form

- Write your model
- Prepare data for the model
- Let the computer translate to & from the solver's form

Limited drawbacks

- Need to learn a new language
- Incur overhead in translation

Great advantages

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

. . . even preferred by programmers

Approaches to Modeling Languages

Algebraic modeling languages

- Designed for “algebraic” formulations
  as seen in our model-based examples
- Excellent fit to many applications and many solvers

Executable approach

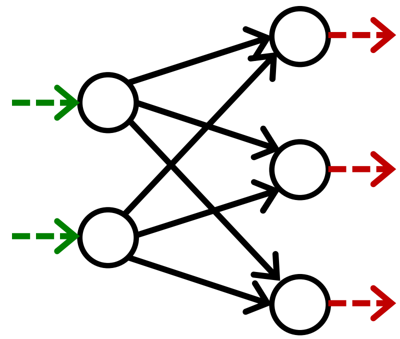
- Write a *computer program* . . .
  - that resembles an optimization model
  - that can be executed to drive a solver

Declarative approach

- Write a *model description* . . .
  - in a language specialized for optimization
  - that can be translated to the solver’s form

Example:

Supply Chain Optimization



Executable approach: *gurobipy*

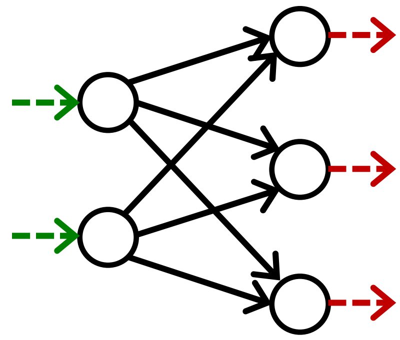
- Based on the Python programming language
  - Designed to look like algebraic notation
- Generates problems for the Gurobi solver

Declarative approach: AMPL

- Based directly on algebraic notation
  - Designed specifically for optimization
- Generates problems for Gurobi and other solvers

Multi-Product Flow

Formulation: Data



Given

P set of products

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_{pj} supply/demand of product p at node j, for each $p \in P$, $j \in N$
> 0 implies supply, < 0 implies demand

c_{pij} cost per unit to ship product p on arc (i,j),
for each $p \in P$, $(i,j) \in A$

Multi-Product Flow

Statements: Data

gurobipy

- Assign values to Python lists and dictionaries

```
products = ['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 PRODUCTS;
set NODES;

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

```
set PRODUCTS := 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  ;
```

Multi-Product Flow

Statements: 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   ;
```

Multi-Product Flow

Statements: 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 }
```

Statements: 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    ;
```

Multi-Product Flow

Formulation: Model

Determine

X_{pij} amount of commodity p to be shipped from node i to node j,
for each $p \in P$, $(i, j) \in A$

to minimize

$\sum_{p \in P} \sum_{(i,j) \in A} c_{pij} X_{pij}$

total cost of shipping

subject to

$\sum_{p \in P} X_{pij} \leq u_{ij}$, for all $(i, j) \in A$

total shipped on each arc must not exceed capacity

$\sum_{(i,j) \in A} X_{pij} + s_{pj} = \sum_{(j,i) \in A} X_{pji}$, for all $p \in P$, $j \in N$

shipments in plus supply/demand must equal shipments out

Multi-Product Flow

Statements: Model

gurobipy

```
m = Model('netflow')

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

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

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

$$\sum_{(i,j)\in A} X_{pij} + s_{pj} = \sum_{(j,i)\in A} X_{pji}, \text{ for all } p \in P, j \in N$$

Multi-Product Flow

Statements: Model

gurobipy

```
m = Model('netflow')

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

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

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

alternatives

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

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

(Note on Summations)

gurobipy quicksum

```
m.addConstrs(
  (quicksum(flow[p,i,j] for i,j in arcs.select('*',j)) + inflow[p,j] ==
   quicksum(flow[p,j,k] for j,k in arcs.select(j,'*'))
      for p 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.

Multi-Product Flow

Statements: Model *(cont'd)*

AMPL

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

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

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

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

$$\sum_{(i,j)\in A} X_{pij} + s_{pj} = \sum_{(j,i)\in A} X_{pji}, \quad \text{for all } p \in P, j \in N$$

Multi-Product Flow

Solution

gurobipy

```
m.optimize()

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

Multi-Product Flow

Solution *(cont'd)*

AMPL

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

ampl: option solver gurobi;
ampl: solve;

Gurobi 9.5.2: optimal solution; objective 5500
1 simplex iteration

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
;
```

Multi-Product Flow

Solution *(cont'd)*

AMPL

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

ampl: option solver xpress;
ampl: solve;

XPRESS 8.11.2(37.01.03): Optimal solution found, Objective 5500
1 simplex iteration

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
;
```

Executable

Advantages

- Complete application development in one environment
- Direct access to advanced solver features

Disadvantages

- Programming and modeling mixed together
  - Model definitions rely on programming concepts
  - Model and data are not separated
  - Modeling and programming bugs are hard to distinguish
- Complications to achieve efficiency

Declarative

Disadvantages

- Adds a new system between application and solver

Advantages

- Focused on optimization modeling
  - Streamlined application prototyping, without programming
  - Faster processing, stronger validation, easier maintenance
- Not specific to one programming language
  - Scripting language extends the model statements with loops, tests, and assignments
  - APIs provide multiple programming language interfaces tailored to C++, C#, Java, MATLAB, Python, R

Integration with Applications

gurobipy

- Everything can be developed in Python
- Part of the Gurobi package
  - Free solver-independent alternatives (Pyomo, PuLP, Python-MIP)

AMPL

- Prototypes can be developed in AMPL
  - Modeling language extended with loops, tests, assignments
- Application programming interfaces (APIs)
  for integrating AMPL with popular programming languages
  - C++, C#, Java, MATLAB, Python, R

Integration with Solvers

gurobipy

- Works closely with the Gurobi solver:
  callbacks during optimization, fast re-solves after problem changes
- Supports Gurobi's 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
  - Connects to global optimization and constraint programming solvers

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
```

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

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

$\sum_{j=1}^{G} x_{ij} = 1,$ 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

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
```

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

Balanced Assignment

Define also

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

$LR_{lr} = \{i \in P\colon t_{i,\mathrm{loc}} = l,\ t_{i,\mathrm{rank}} = r\}$, for all $l \in T_{\mathrm{loc}}, r \in T_{\mathrm{rank}}$

$A_r \subseteq T_{\mathrm{rank}}$, for all $r \in T_{\mathrm{rank}}$

Subject to also

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

no group may have only one woman assigned

$\sum_{i \in LR_{lr}} x_{ij} = 0$ or $\sum_{i \in LR_{lr}} x_{ij} + \sum_{a \in A_r} \sum_{i \in LR_{la}} x_{ij} \geq 2$,

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

for each person in each location, there must be
at least one other person of the same or an adjacent rank

Balanced Assignment *in AMPL*

Complicating constraints

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

subj to Min2WomenInGroupLO {j in 1..numberGrps}:
   sum {i in WOMEN} Assign[i,j] = 0 or sum {i in WOMEN} Assign[i,j] >= 2;
```

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

```
set LOCRANK {l in TYPES['loc'], r in TYPES['rank']} =
   {i in PEOPLE: type[i,'loc'] = l and type[i,'rank'] = r};

set ADJACENT {r in TYPES['rank']} within TYPES['rank'] diff {r};

subj to NoPersonIsolated
         {l in TYPES['loc'], r in TYPES['rank'], j in 1..numberGrps}:
   sum {i in LOCRANK[l,r]} Assign[i,j] = 0 or
   sum {i in LOCRANK[l,r]} Assign[i,j] +
      sum {a in ADJACENT[r]} sum {i in LOCRANK[l,a]} Assign[i,j] >= 2;
```

New!

Extended Solver Interface

Supported operators and functions

- Conditional: `if-then-else`; `==>`, `<==`, `<==>`
- Logical: `or`, `and`, `not`; `exists`, `forall`
- Piecewise linear: `abs`; `min`, `max`; `<<breakpoints; slopes>>`
- Counting: `count`; `atmost`, `atleast`, `exactly`; `numberof`
- Comparison: `>`, `<`, `!=`; `alldiff`
- Complementarity: `complements`
- Nonlinear: `*`, `/`, `^`; `exp`, `log`; `sin`, `cos`, `tan`; `sinh`, `cosh`, `tanh`
- Set membership: `in`

Supported solvers

- Gurobi, COPT, HiGHS, *Xpress coming soon, . . .*

Modeling guide

- *https://amplmp.readthedocs.io/en/latest/rst/model-guide.html*

Modeling Language Data

210 people, 4 categories

- 18 types, 12 groups, 16-19 people/group

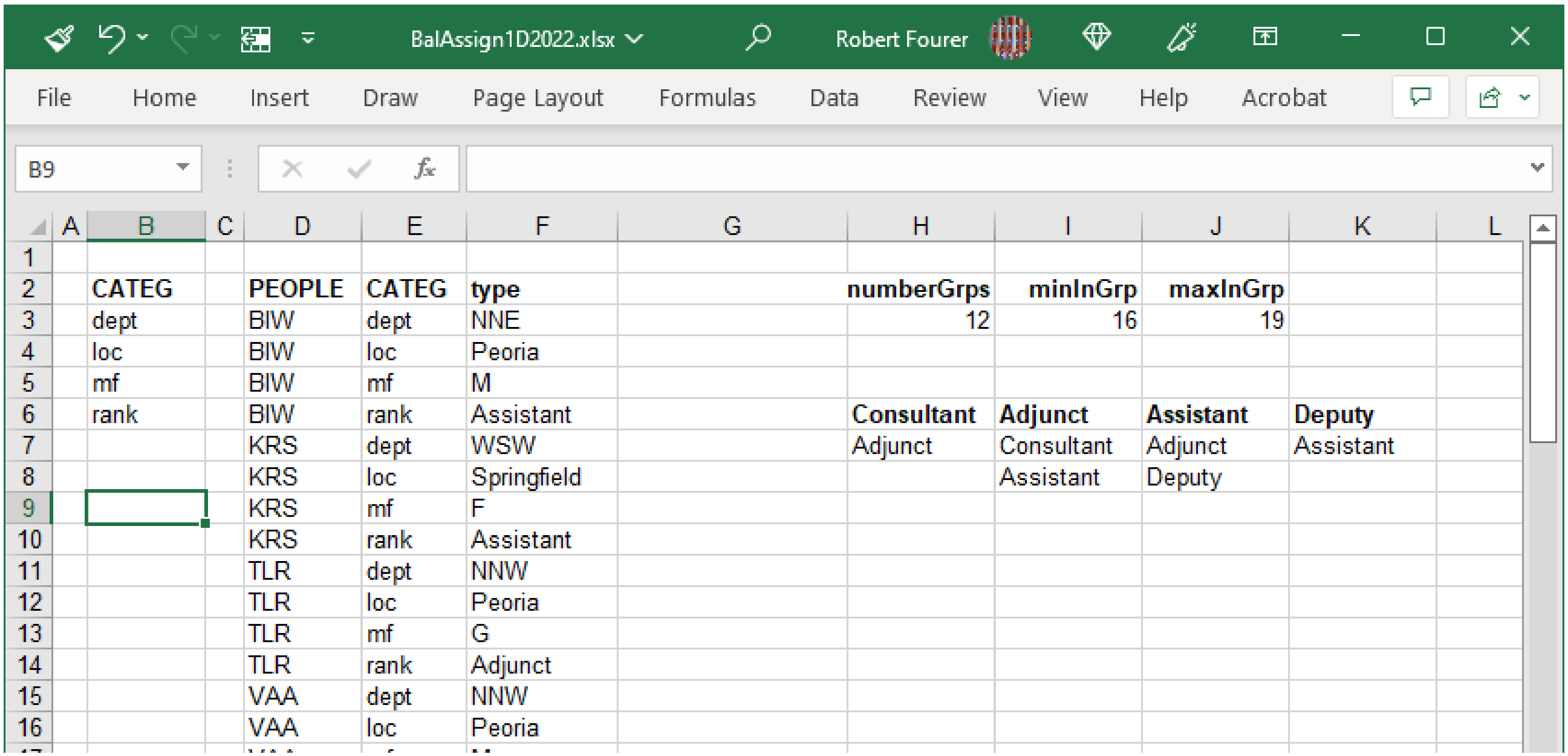
```
set PEOPLE :=
   BIW   AJH   FWI   IGN   KWR   KKI   HMN   SML   RSR   TBR
   KRS   CAE   MPO   CAR   PSL   BCG   DJA   AJT   JPY   HWG
   TLR   MRL   JDS   JAE   TEN   MKA   NMA   PAS   DLD   SCG
.......

set CATEG := dept loc 'm-f' rank ;

param type:
     dept        loc      'm-f'   rank   :=
BIW   NNE   Peoria         M   Assistant
KRS   WSW   Springfield    F   Assistant
TLR   NNW   Peoria         F   Adjunct
VAA   NNW   Peoria         M   Deputy
.......

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

New!

Direct Spreadsheet Table Interface

One-dimensional listing

- `param type {PEOPLE,CATEG} symbolic;`



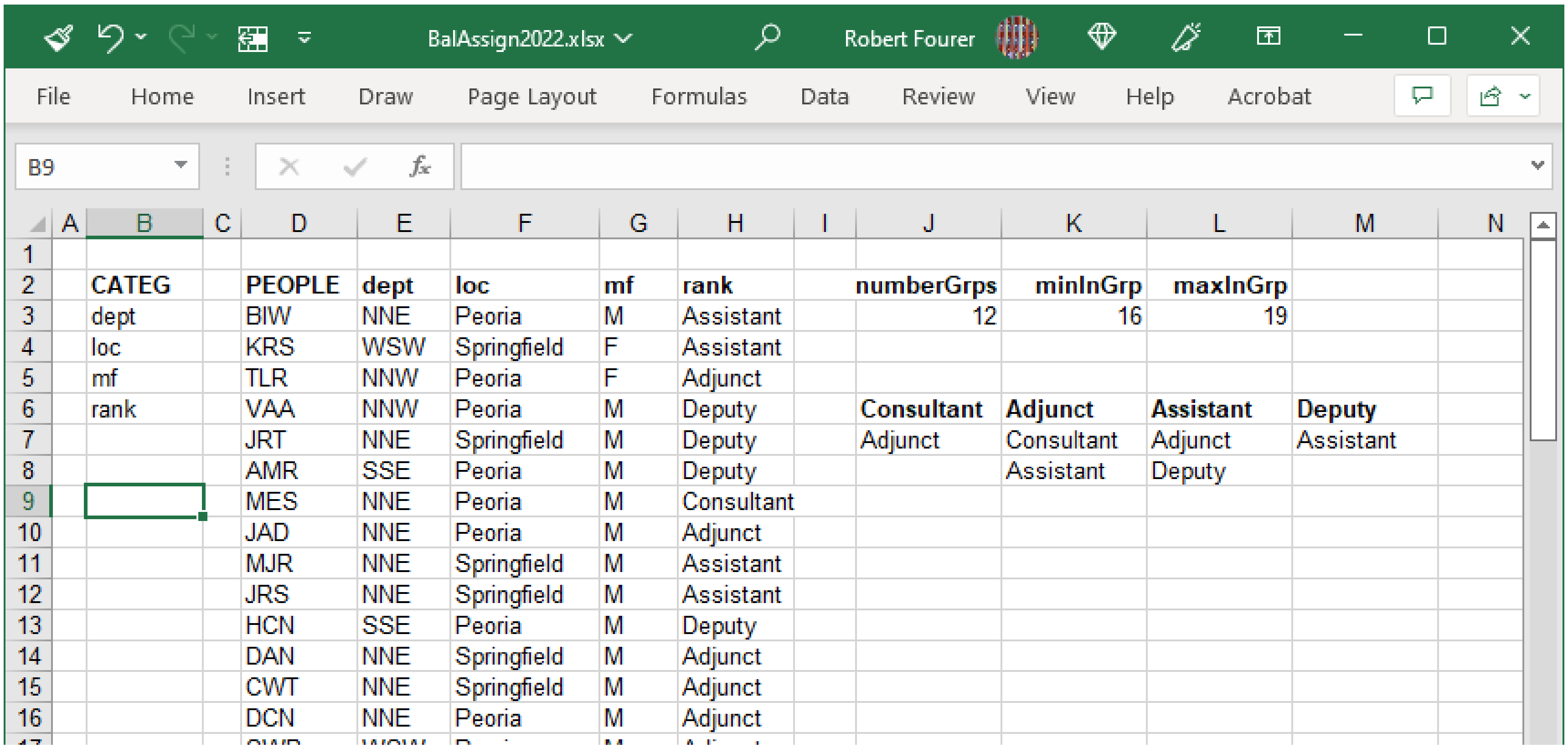
	A	B	C	D	E	F	G	H	I	J	K	L
1												
2		**CATEG**		**PEOPLE**	**CATEG**	**type**		**numberGrps**	**minInGrp**	**maxInGrp**		
3		dept		BIW	dept	NNE		12	16	19		
4		loc		BIW	loc	Peoria						
5		mf		BIW	mf	M						
6		rank		BIW	rank	Assistant		**Consultant**	**Adjunct**	**Assistant**	**Deputy**	
7				KRS	dept	WSW		Adjunct	Consultant	Adjunct	Assistant	
8				KRS	loc	Springfield			Assistant	Deputy		
9				KRS	mf	F						
10				KRS	rank	Assistant						
11				TLR	dept	NNW						
12				TLR	loc	Peoria						
13				TLR	mf	G						
14				TLR	rank	Adjunct						
15				VAA	dept	NNW						
16				VAA	loc	Peoria						

New!

Direct Spreadsheet Table Interface

Two-dimensional table

- `param type {PEOPLE,CATEG} symbolic;`



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2		**CATEG**		**PEOPLE**	**dept**	**loc**	**mf**	**rank**		**numberGrps**	**minInGrp**	**maxInGrp**		
3		dept		BIW	NNE	Peoria	M	Assistant		12	16	19		
4		loc		KRS	WSW	Springfield	F	Assistant						
5		mf		TLR	NNW	Peoria	F	Adjunct						
6		rank		VAA	NNW	Peoria	M	Deputy		**Consultant**	**Adjunct**	**Assistant**	**Deputy**	
7				JRT	NNE	Springfield	M	Deputy		Adjunct	Consultant	Adjunct	Assistant	
8				AMR	SSE	Peoria	M	Deputy			Assistant	Deputy		
9				MES	NNE	Peoria	M	Consultant						
10				JAD	NNE	Peoria	M	Adjunct						
11				MJR	NNE	Springfield	M	Assistant						
12				JRS	NNE	Springfield	M	Assistant						
13				HCN	SSE	Peoria	M	Deputy						
14				DAN	NNE	Springfield	M	Adjunct						
15				CWT	NNE	Springfield	M	Adjunct						
16				DCN	NNE	Peoria	M	Adjunct						

Balanced Assignment

Modeling Language Script

Read model & data, solve, write solution

```
model BalAssign2022.mod;

table Categories IN "amplxl" "bal.xlsx": CATEG <- [CATEG];
table People IN "amplxl" "bal.xlsx": PEOPLE <- [PEOPLE];
table Types IN "amplxl" "bal.xlsx" "2D": [PEOPLE,CATEG], type;
table Groups IN "amplxl" "bal.xlsx": [], numberGrps, minInGrp, maxInGrp;
table Adjacent {r in TYPES['rank’]}
   IN "amplxl" "bal.xlsx": ADJACENT[r] <- [(r)];
read table Categories; read table People;
read table Types; read table Groups; read table Adjacent;

option solver x-gurobi;
solve;

table Summary {k in CATEG} OUT "amplxl" "bal.xlsx" (k) "2D":
   {j in 1..numberGrps, l in TYPES[k]} -> [Group,Type],
      sum {i in PEOPLE: type[i,k] = l} Assign[i,j];
write table Summary;
```

Modeling Language Execution

Load spreadsheet handler, execute script

```
ampl: load amplxl.dll;

ampl: include BalAssign2022.run;

Presolve eliminates 72 arithmetic and 144 logical constraints.

Adjusted problem:
2556 variables:
        2520 nonlinear variables
        36 linear variables
582 algebraic constraints, all linear; 25224 nonzeros
        210 equality constraints
        360 inequality constraints
        12 range constraints
252 logical constraints
1 linear objective; 2 nonzeros.

x-Gurobi 9.5.1: optimal solution; objective 25

134242 simplex iterations
816 branching nodes
```

50.4 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 arithmetic and 144 logical constraints.
...

Gurobi 9.5.1: optimal solution; objective 25
158275 simplex iterations
851 branch-and-cut nodes
```

21.9 sec

Balanced Assignment

Modeling Language Results

Rank

Group	Assistant	Adjunct	Deputy	Consultant
1	8	7	2	0
2	8	7	2	1
3	8	8	2	1
4	7	7	2	1
5	8	8	1	0
6	7	7	2	1
7	8	8	2	0
8	7	8	2	1
9	7	8	2	0
10	8	8	2	0
11	7	7	2	1
12	8	7	2	0

Balanced Assignment

Modeling Language Results

Location

Group	Peoria	Springfield	Macomb	Urbana	Joliet	Carbondale	Cairo	Evansville
1	11	4	0	0	2	0	0	0
2	10	4	0	0	2	0	0	2
3	10	4	2	3	0	0	0	0
4	10	4	3	0	0	0	0	0
5	11	4	0	0	0	2	0	0
6	11	4	0	0	0	2	0	0
7	10	4	0	0	2	2	0	0
8	10	4	0	0	2	0	2	0
9	10	5	0	0	0	2	0	0
10	10	4	0	0	2	2	0	0
11	11	4	0	0	0	2	0	0
12	11	4	0	0	0	2	0	0

Solvers for Model-Based Optimization

Ready-to-run solvers for broad problem classes

Three widely used types

- “Linear”
- “Nonlinear”
- “Global”

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

“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 linear:
    piecewise-linear expressions, logic in constraints, . . .

Quadratic extensions

- Convex elliptic objectives and constraints
- Convex conic constraints
- $x_j u_j$ terms, where u_j is a zero-one variable
- General non-convex quadratic expressions

“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

Require function and derivative evaluations

Continuous variables, local optimality

- Smooth objective and constraint functions
  - *Derivative computations handled by callbacks to AMPL interface*
- Variety of methods
  - Interior-point, sequential quadratic, reduced gradient

Some extend 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 expressions, global optimality

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

“Global” Solvers

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

Try AMPL!

New! Free AMPL Community Edition *ampl.com/ce*

- Free AMPL and open-source solvers
  - no size or time limitations
- 1-month full-featured trials of commercial solvers
- Requires internet connection for validation

Time-limited trials, size-limited demos *ampl.com/try-ampl*

Free AMPL for Courses *ampl.com/try-ampl/ampl-for-courses*

- Full-featured, time limited

Free AMPL web access

- AMPL model colaboratory *colab.ampl.com*
- NEOS Server *neos-server.org*