Prescriptive Analytics with AMPL

building optimization applications quickly and reliably, from prototyping to deployment

Filipe Brandão, Robert Fourer

`{fdabrandao,4er}@ampl.com`

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

INFORMS Business Analytics Conference

Aurora, Colorado — 16 April 2023
Technology Workshop

Prescriptive Analytics with AMPL: building optimization applications quickly and reliably, from prototyping to deployment

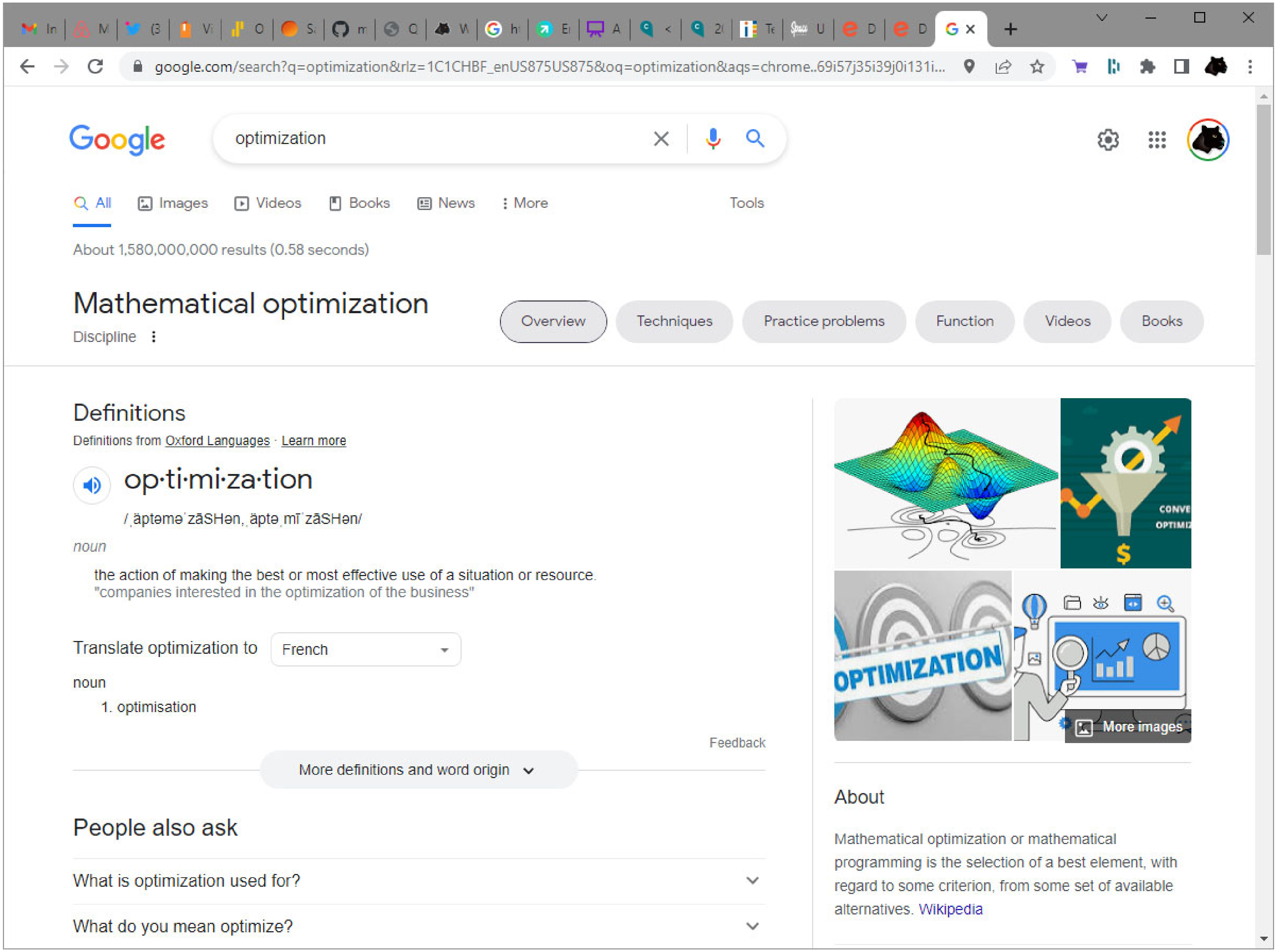
Optimization is the most widely adopted technology of Prescriptive Analytics, but also the most challenging to implement. This presentation takes you through the steps of a proven approach that combines the best features of two implementation environments:

- Prototyping in Google Colab using AMPL, a language and system designed for the needs of formulating and validating optimization models
- Deployment using Python-based tools, the most popular environment for building Analytics models into deployable applications

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 specialization to model-based optimization is able to offer exceptional power of expression and speed of execution while maintaining ease of use. Recent enhancements to the AMPL language let you write many common logical conditions in an even more natural way, avoiding complicated reformulations. To support the prototyping phase, expanded data handlers facilitate direct import of values in spreadsheet, CSV, JSON, and database formats.

Our presentation next shows how AMPL and Python work together for building optimization into enterprise systems. AMPL fits naturally into the Python framework, installing as an "amplpy" Python package, importing and exporting data naturally from/to Python data structures and Pandas dataframes, and supporting Jupyter notebooks that mix AMPL modeling and Python programming. In contrast to Python-only modeling solutions, AMPL's Python API offers straightforward, efficient model processing while leveraging Python's vast ecosystem for data pre-processing, solution analysis, and visualization.

We finish with a deployment example, showing how Python scripts can be turned quickly into Prescriptive Analytics applications using amplpy, Pandas, and the Streamlit app framework. Deployments are supported on traditional servers and in a variety of modern virtual environments including containers, clusters, and cloud machines.


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

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

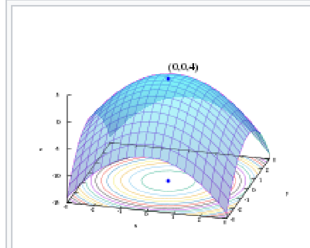
Mathematical optimization

From Wikipedia, the free encyclopedia

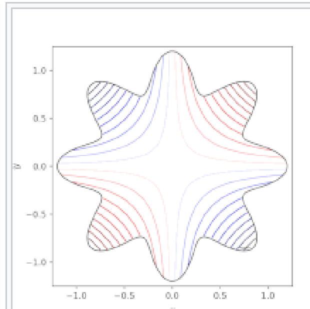
"Mathematical programming" redirects here. For the peer-reviewed journal, see Mathematical Programming.
"Optimization" and "Optimum" redirect here. For other uses, see Optimization (disambiguation) and Optimum (disambiguation).

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 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

Part 1: Development & Prototyping

AMPL's approach

- Optimization: *model-based not method-based*
- Model-based optimization:
  a modeling language not a programming language
- Modeling language: *declarative not executable*

New and notable in AMPL

- Formulate models more like you think about them
  - Choose from the best ready-to-run solvers
- Read & write essential data formats
- Choose your preferred working environment
  - *Get started with Python notebooks and Google Colab . . .*

Part 2: AMPL Integration & Deployment

Part 2: Python Integration & Deployment

AMPL's approach

- ❖ Fits naturally into the *Python ecosystem*
- ❖ Works with *Python data structures*
- ❖ Leverages *Python libraries and tools*

New and notable in AMPL

- ❖ Python packages for AMPL & all solvers
- ❖ amplpy (AMPL Python API)
  - ∗ natural integration with Pandas dataframes
- ❖ Snapshots of models & data
- ❖ Fast app building with Streamlit & Streamlit Community Cloud
- ❖ Virtual environments on any cloud platform
  - ∗ Docker containers
  - ∗ Kubernetes clusters & cloud functions
  - ∗ continuous integration (CI/CD)
- ❖ Dynamic licensing system and usage dashboard

Example:

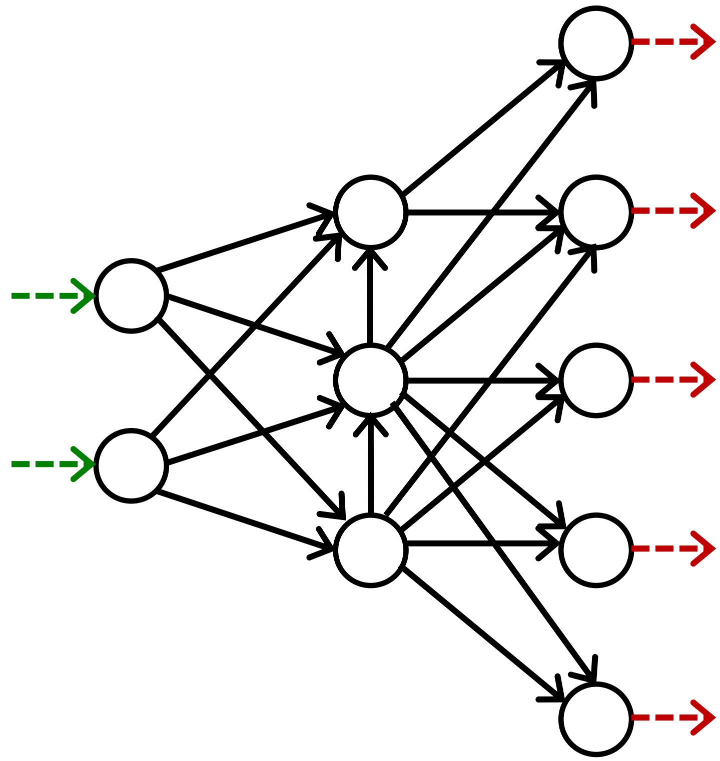
Multi-Product Optimal Network Flow

Motivation

- Ship products efficiently
  to meet demands

Context

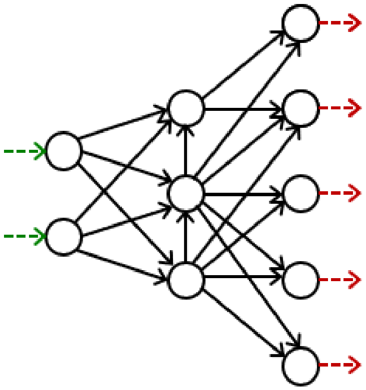
- a transportation network
  - ∗ nodes ◯ representing cities
  - ∗ arcs ⟶ representing roads
- supplies - - -> at nodes
- demands - - -> at nodes
- capacities on arcs
- shipping costs on arcs



Multi-Product Network Flow

Decide

- how much of each product to ship on each arc



So that

- shipping costs are kept low
- shipments on each arc respect capacity of the arc
- supplies, demands, and shipments are in balance at each node

Approaches to Mathematical Optimization

Method-based approach

- Program a method to build a shipping plan

Model-based approach

- Formulate a minimum shipping cost model

Method-Based Approach

Program a method to build a shipping plan

- “method”: says how to compute a solution

Order-driven method?

- Develop rules for how each order should be met
  - Given some demand and given available capacity, determine where to ship it from and which route to use
- Fill orders one by one, according to the rules
  - Decrement capacity as each one is filled

Route-driven method?

- Repeat until all demands are met
  - Choose a shipping route and a product
  - Add as much flow as possible of that product along that route without exceeding supply, demand, or capacity

Method-Based Refinements

Develop rules for choosing good routes

- Generate batches of routes
- Apply routes in some systematic order

Improve the initial solution

- *Local optimization:* swaps and other simple improvements
- *Local-search metaheuristics:*
  simulated annealing, tabu search, GRASP
- *Population-based metaheuristics:*
  evolutionary methods, particle swarm optimization

Model-Based Approach

Formulate a minimum shipping cost model

- “model”: says what a solution should satisfy
- Identify amounts shipped
  as the decisions of the model *(variables)*
- Specify feasible shipment amounts
  by writing equations that the variables must satisfy *(constraints)*
- Write total shipping cost
  as a summation over the variables *(objective)*
- Collect costs, capacities, supplies, demands *(data)*

Send to a solver that computes optimal solutions

- Available as a ready-to-run package
- Handles entire mathematical problem classes efficiently
  - linear constraints and objective, continuous and integer variables
- Recognizes and exploits special cases

Multi-Product Flow

Model-Based Formulation

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

Model-Based Formulation *(cont'd)*

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

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

on each arc, total shipped must not exceed capacity

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

at each node, shipments in plus
supply/demand must equal shipments out

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

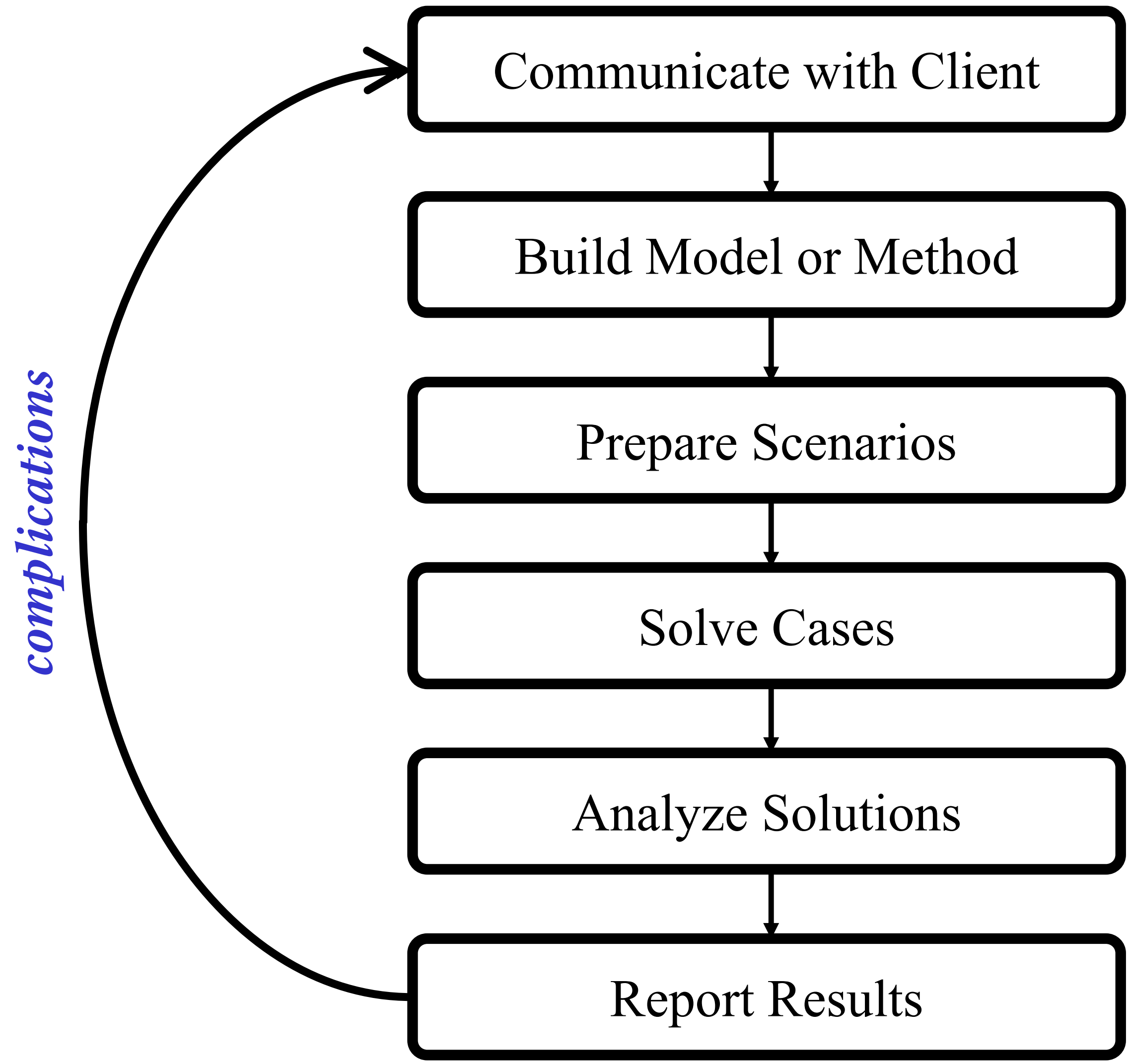
Where is the effort?

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

Which should you prefer?

- For simple problems, either approach can seem workable
- *But in real optimization there are always* ***complications*** *. . .*

The Optimization Application Cycle



The Optimization Application Cycle

Goals for optimization modelers

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

Goals for optimization software

- Fast prototyping
- Easy integration for deployment
- Successful long-term maintenance

Example:

Complications in Multi-Product Flow

Additional restrictions imposed by the client

- Cost has fixed and variable parts
  - Each arc incurs a cost if it is *used* for shipping
- Shipments cannot be too small
- Not too many arcs can be used

Additional data for the problem

d_{ij} fixed cost for using the arc from i to j, for each $(i,j) \in A$

m smallest total that may be shipped on any arc used

n largest number of arcs that may be used

Method-Based *(cont'd)*

What has to be done?

- Revise or re-think the solution approach
- Update or re-implement the algorithm

What are the challenges?

- In this example,
  - Shipments have become more interdependent
  - Good routes are harder to identify
  - Improvements are harder to find
- In general,
  - Even small changes to a problem can necessitate major changes to the method and its implementation
  - Each problem change requires more method development

. . . and problem changes are frequent!

Model-Based *(cont'd)*

What has to be done?

- Update the objective expression
- Formulate additional constraint equations
- Send back to the solver

What are the challenges?

- In this example,
  - New variables and expressions to represent fixed costs
  - New constraints to impose shipment and arc-use limits
- In general,
  - The formulation tends to get more complicated
  - A new solver type or solver options may be needed

. . . but it's easier to update formulations than methods
. . . and a few solver types handle most cases

Model-Based Formulation *(revised)*

Determine

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

Y_{ij} 1 if any amount is shipped from node i to node j, 0 otherwise, for each $(i, j) \in A$

to minimize

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

total cost of shipments

Model-Based Formulation *(revised)*

Subject to

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

when the arc from node i to node j is used for shipping, total shipments must not exceed capacity, and Y_{ij} must be 1

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

shipments in plus supply/demand must equal shipments out

$\sum_{p\in P} X_{pij} \ge m Y_{ij}$, for all $(i,j) \in A$

when the arc from node i to node j is used for shipping, total shipments from i to j must be at least m

$\sum_{(i,j)\in A} Y_{ij} \le n$

At most n arcs can be used

Method-Based Use Cases . . .

Your problem is too 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
- Logistics, air/rail/truck operations, 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

- Operations research 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

Approaches to Model-Based Optimization

Translate between two forms of the problem

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

Programming language approach

- Write a program to generate the solver's form

Modeling language approach

- Write the model formulation
  in a language 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:

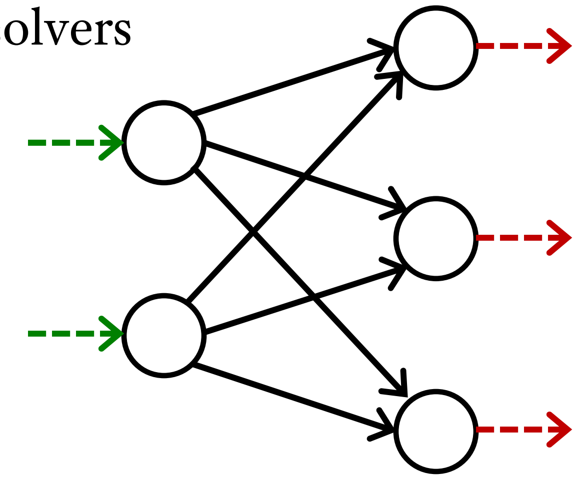
Multi-Product Optimal Network Flow

Executable approach: *gurobipy*

- Based on the Python programming language
- Generates problems for the Gurobi solver

Declarative approach: AMPL

- Based on algebraic notation (like our sample formulation)
- 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 $p \in P$, $(i, j) \in A$

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 }
```

Multi-Product Flow

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")
```

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

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 highs;
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
;
```

Multi-Product Flow

Solution *(cont'd)*

AMPL

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

ampl: option solver highs;
ampl: solve;

HiGHS 1.4.0: optimal solution; objective 5500
1 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
;
```

Approaches to Complications

Easily accommodated

- Add variables to the model
- Add a term to the objective
- Update and/or add constraints
- Send to the same solver

Mixed-integer approach

- Use zero-one variables
  to express the logic of the complicating constraints

Direct logic approach

- Write the logic of the complicating constraints
  directly using an enhanced modeling language

Multi-Product Flow

Mixed-Integer Model in AMPL

Symbolic data, variables, objective

```
set PRODUCTS;
set NODES;

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

param inflow {PRODUCTS,NODES};
param min_ship >= 0;
param max_arcs >= 0;

param var_cost {PRODUCTS,ARCS} >= 0;
var Flow {PRODUCTS,ARCS} >= 0;

param fix_cost {ARCS} >= 0;
var Use {ARCS} binary;

minimize TotalCost:
   sum {p in PRODUCTS, (i,j) in ARCS} var_cost[p,i,j] * Flow[p,i,j] +
   sum {(i,j) in ARCS} fix_cost[i,j] * Use[i,j];
```

Multi-Product Flow

Mixed-Integer Model *(cont'd)*

Constraints

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

subject to Min_Shipment {(i,j) in ARCS}:
   sum {p in PRODUCTS} Flow[p,i,j] >= min_ship * Use[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];

subject to Max_Used:
   sum {(i,j) in ARCS} Use[i,j] <= max_arcs;
```

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

Multi-Product Flow

Data Instance in AMPL Text Format

Limits

```
set PRODUCTS := Bands Coils ;
set NODES := Detroit Denver Boston 'New York' Seattle ;

param: ARCS: capacity:

         Boston 'New York' Seattle :=
  Detroit   100       80       120
  Denver    120      120       120   ;

param inflow:
         Detroit Denver Boston 'New York' Seattle :=
  Bands      50      60     -50      -50        -10
  Coils      60      40     -40      -30        -30;

param min_ship := 15 ;

param max_arcs := 4 ;
```

Multi-Product Flow

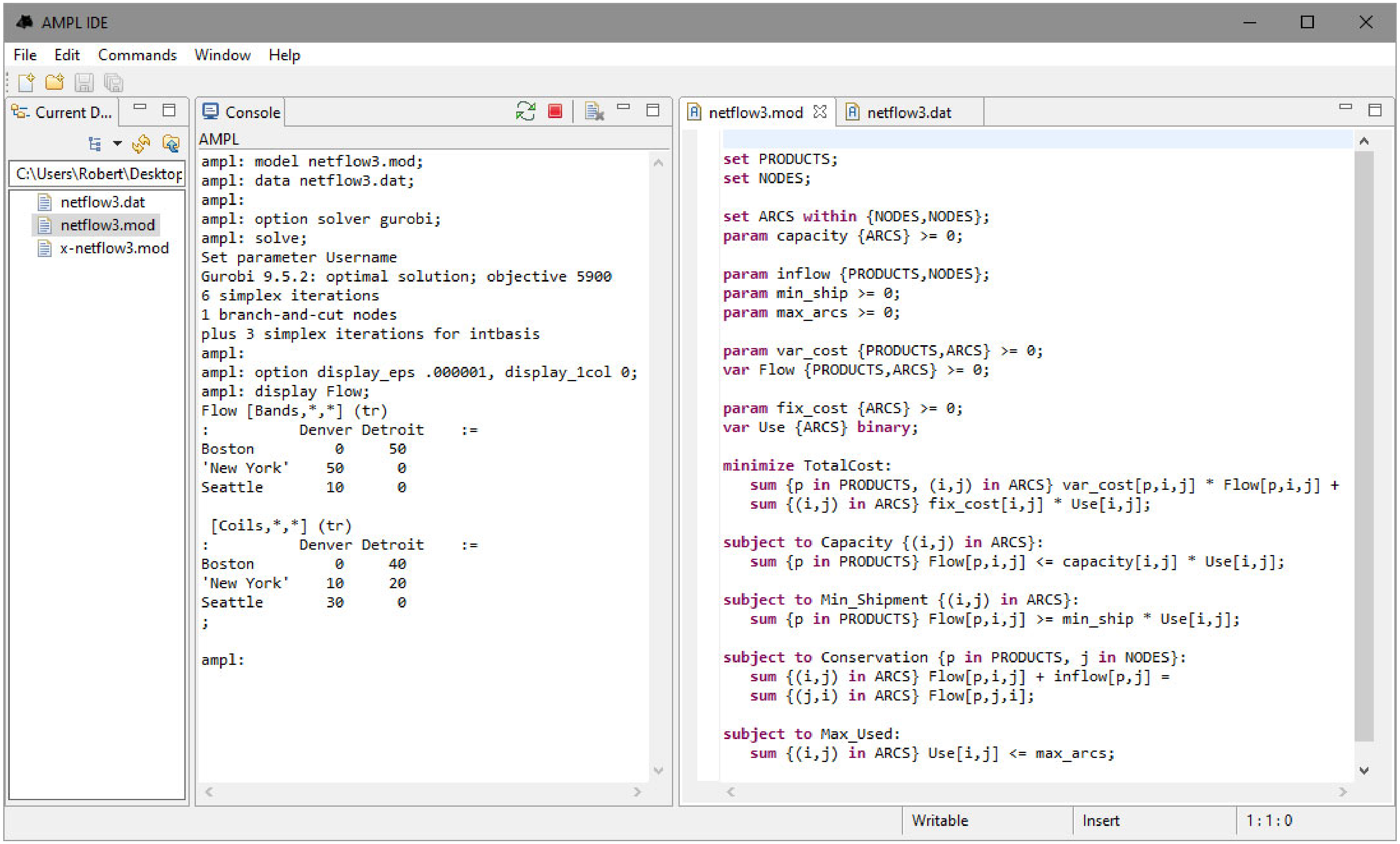
Data Instance *(cont'd)*

Costs

```
param var_cost:
 [Bands,*,*]:  Boston 'New York' Seattle :=
     Detroit     10        20        60
     Denver      40        40        30
 [Coils,*,*]:  Boston 'New York' Seattle :=
     Detroit     20        20        80
     Denver      60        70        30 ;

param fix_cost default 75 ;
```

Multi-Product Flow

Optimization by the Gurobi Solver



Multi-Product Flow

Direct Logic Model in AMPL

Symbolic data, variables, objective

```
set PRODUCTS;
set NODES;

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

param inflow {PRODUCTS,NODES};
param min_ship >= 0;
param max_arcs >= 0;

param fix_cost {ARCS} >= 0;
param var_cost {PRODUCTS,ARCS} >= 0;

var Flow {PRODUCTS,ARCS} >= 0;

minimize TotalCost:
   sum {p in PRODUCTS, (i,j) in ARCS} var_cost[p,i,j] * Flow[p,i,j] +
   sum {(i,j) in ARCS}
      if exists {p in PRODUCTS} Flow[p,i,j] > 0 then fix_cost[i,j];
```

Multi-Product Flow

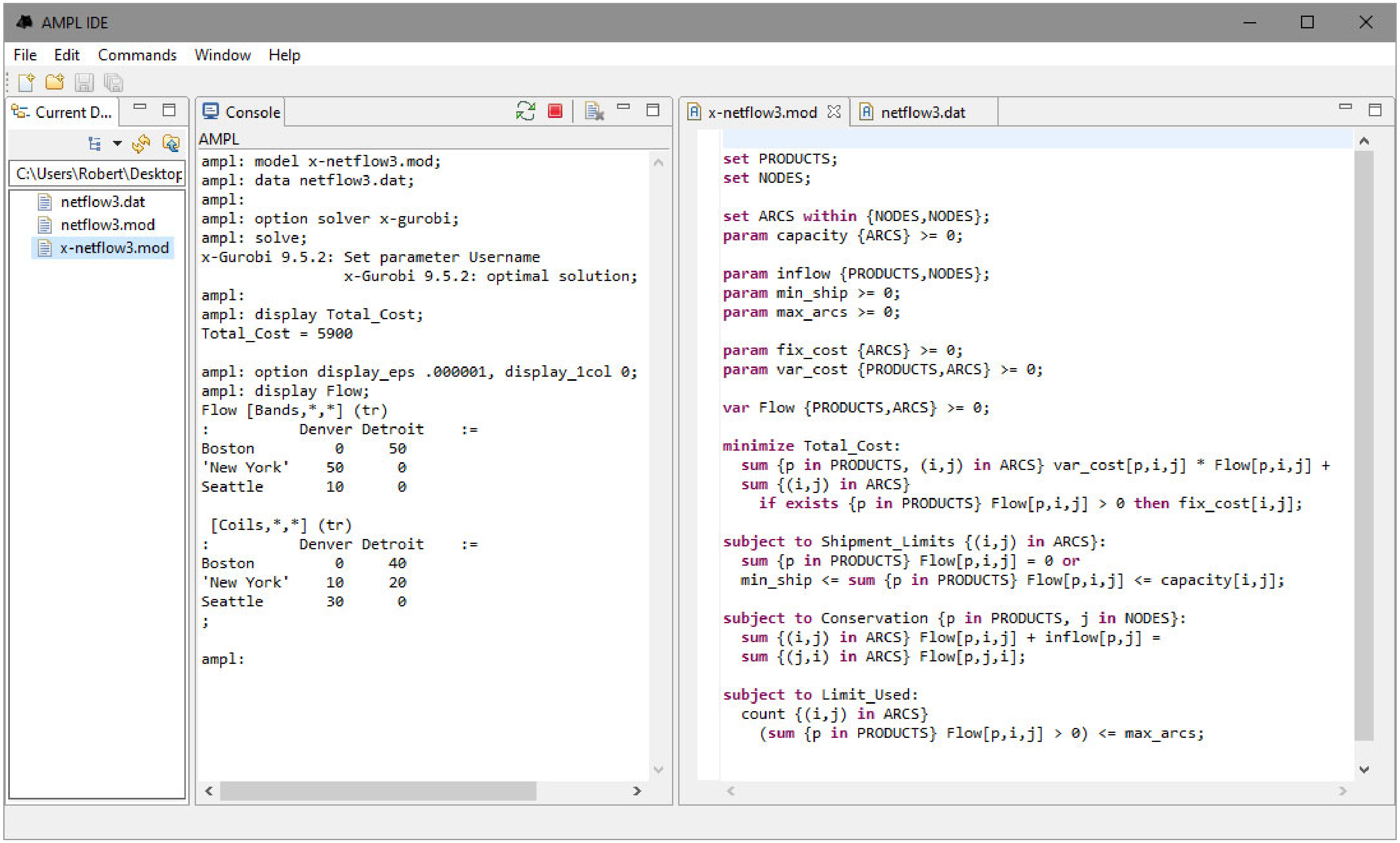
Direct Logic Model in AMPL

Constraints

```
subject to Capacity {(i,j) in ARCS}:
   sum {p in PRODUCTS} Flow[p,i,j] = 0 or
   min_ship <= 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];

subject to Max_Used:
   count {(i,j) in ARCS}
      (sum {p in PRODUCTS} Flow[p,i,j] > 0) <= max_arcs;
```

Multi-Product Flow

Optimization by the Gurobi Solver




```
ampl: model x-netflow3.mod;
ampl: data netflow3.dat;
ampl:
ampl: option solver x-gurobi;
ampl: solve;
x-Gurobi 9.5.2: Set parameter Username
                x-Gurobi 9.5.2: optimal solution;
ampl:
ampl: display Total_Cost;
Total_Cost = 5900

ampl: option display_eps .000001, display_1col 0;
ampl: display Flow;
Flow [Bands,*,*] (tr)
:          Denver Detroit    :=
Boston        0     50
'New York'   50      0
Seattle      10      0

 [Coils,*,*] (tr)
:          Denver Detroit    :=
Boston        0     40
'New York'   10     20
Seattle      30      0
;

ampl:
```

```
set PRODUCTS;
set NODES;

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

param inflow {PRODUCTS,NODES};
param min_ship >= 0;
param max_arcs >= 0;

param fix_cost {ARCS} >= 0;
param var_cost {PRODUCTS,ARCS} >= 0;

var Flow {PRODUCTS,ARCS} >= 0;

minimize Total_Cost:
  sum {p in PRODUCTS, (i,j) in ARCS} var_cost[p,i,j] * Flow[p,i,j] +
  sum {(i,j) in ARCS}
    if exists {p in PRODUCTS} Flow[p,i,j] > 0 then fix_cost[i,j];

subject to Shipment_Limits {(i,j) in ARCS}:
  sum {p in PRODUCTS} Flow[p,i,j] = 0 or
  min_ship <= 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];

subject to Limit_Used:
  count {(i,j) in ARCS}
    (sum {p in PRODUCTS} Flow[p,i,j] > 0) <= max_arcs;
```

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, Xpress, MOSEK, COPT, HiGHS, CBC, *CPLEX coming* . . .

Modeling guide

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

Importing Data and Exporting Solutions

New data file interfaces

- Spreadsheet files
  - Works with all .xlsx files on Windows, Linux, macOS
  - Supports "two-dimensional" spreadsheet tables
- Comma-separated value (.csv) files
- JSON files

New ODBC interface for database systems

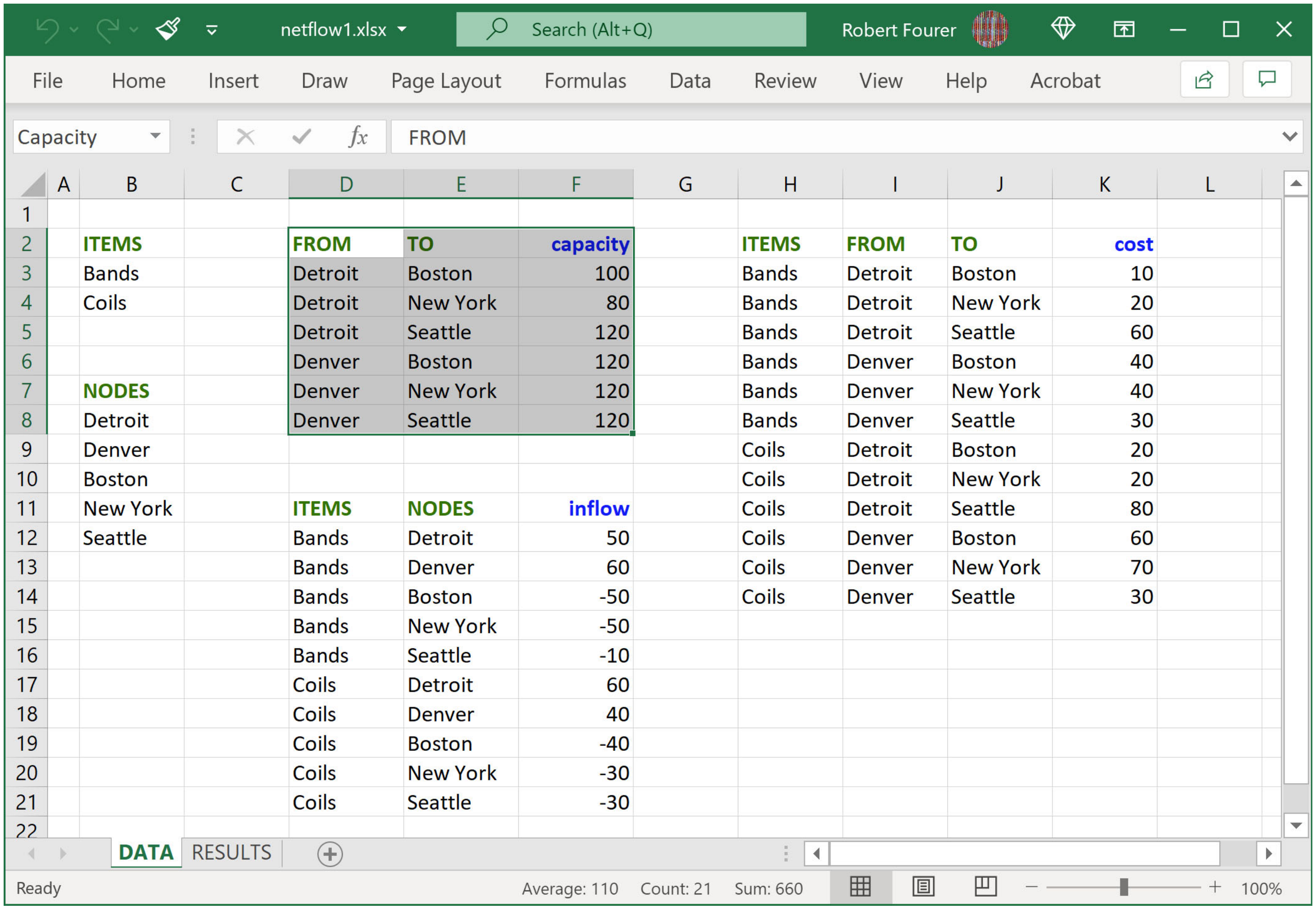
- Support for Microsoft ODBC, unixODBC, iODBC
- Faster operation
- Extended update features

Python API

- Python-AMPL data interchange methods
- Support for lists, sets, tuples, dictionaries, and Pandas

Direct Spreadsheet Interface

“1D” spreadsheet ranges



ITEMS
Bands
Coils

NODES
Detroit
Denver
Boston
New York
Seattle

FROM	TO	capacity
Detroit	Boston	100
Detroit	New York	80
Detroit	Seattle	120
Denver	Boston	120
Denver	New York	120
Denver	Seattle	120

ITEMS	NODES	inflow
Bands	Detroit	50
Bands	Denver	60
Bands	Boston	-50
Bands	New York	-50
Bands	Seattle	-10
Coils	Detroit	60
Coils	Denver	40
Coils	Boston	-40
Coils	New York	-30
Coils	Seattle	-30

ITEMS	FROM	TO	cost
Bands	Detroit	Boston	10
Bands	Detroit	New York	20
Bands	Detroit	Seattle	60
Bands	Denver	Boston	40
Bands	Denver	New York	40
Bands	Denver	Seattle	30
Coils	Detroit	Boston	20
Coils	Detroit	New York	20
Coils	Detroit	Seattle	80
Coils	Denver	Boston	60
Coils	Denver	New York	70
Coils	Denver	Seattle	30

Direct spreadsheet interface

Data Handling

Script (input)

```
model netflow1.mod;

table Products IN "amplxl" "netflow2.xlsx" "Items":
    PRODUCTS <- [ITEMS];

table Nodes IN "amplxl" "netflow2.xlsx":
    NODES <- [NODES];

table Capacity IN "amplxl" "netflow2.xlsx":
    ARCS <- [FROM,TO], capacity;

table Inflow IN "amplxl" "netflow2.xlsx":
    [ITEMS,NODES], inflow;

table Cost IN "amplxl" "netflow2.xlsx":
    [ITEMS,FROM,TO], cost;

load amplxl.dll;

read table Products; read table Nodes;
read table Capacity; read table Inflow; read table Cost;
```

Direct spreadsheet interface

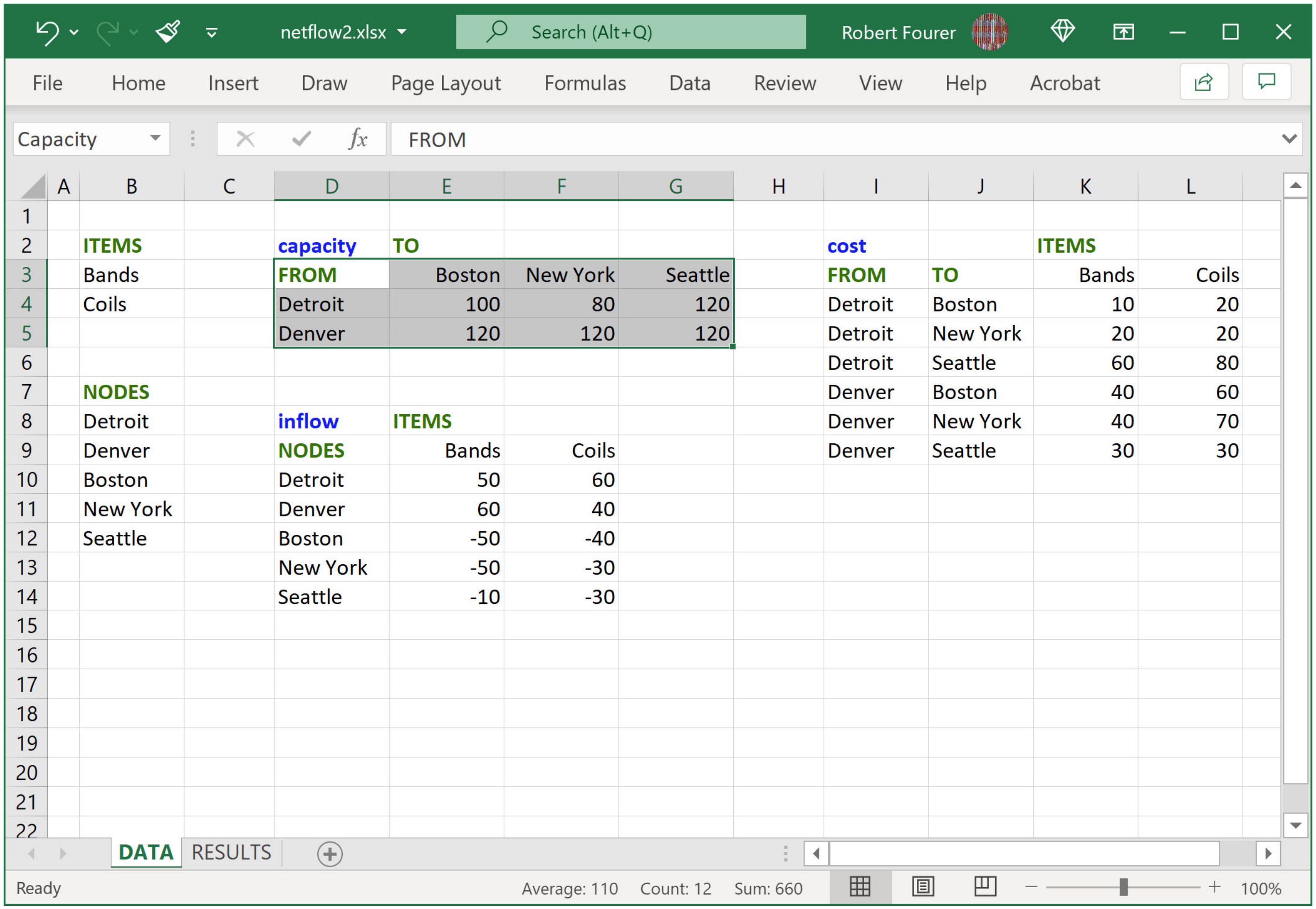
Data Handling

Script (input)

```
model netflow1.mod;

table Products IN "amplxl" "netflow2.xlsx" "Items":
    PRODUCTS <- [ITEMS];

table Nodes IN "amplxl" "netflow2.xlsx":
    NODES <- [NODES];

table Capacity IN "amplxl" "netflow2.xlsx" "2D":
    ARCS <- [FROM,TO], capacity;

table Inflow IN "amplxl" "netflow2.xlsx" "2D":
    [ITEMS,NODES], inflow;

table Cost IN "amplxl" "netflow2.xlsx" "2D":
    [ITEMS,FROM,TO], cost;

load amplxl.dll;

read table Products; read table Nodes;
read table Capacity; read table Inflow; read table Cost;
```

Direct Spreadsheet Interface

“2D” spreadsheet ranges



Direct spreadsheet interface

Data Handling

Script (output)

```
option solver gurobi;
solve;

table Results OUT "amplxl" "netflow1.xlsx" "2D":
    [ITEMS,FROM,TO], Flow;

table Summary OUT "amplxl" "netflow1.xlsx":
    {(i,j) in ARCS} -> [FROM,TO],
    sum {p in PRODUCTS} Flow[p,i,j] ~ TotFlow,
    sum {p in PRODUCTS} Flow[p,i,j] / capacity[i,j] ~ "%Used";

write table Results;
write table Summary;
```

Direct spreadsheet interface

Data Results

“2D” spreadsheet range

netflow2.xlsx

	A	B	C	D	E	F	G	H	I	J
1										
2		shipments		ITEMS						
3		FROM	TO	Bands	Coils		FROM	TO	TotFlow	%Used
4		Detroit	Boston	50	30		Detroit	Boston	80	80.0%
5		Detroit	New York	0	30		Detroit	New York	30	37.5%
6		Detroit	Seattle	0	0		Detroit	Seattle	0	0.0%
7		Denver	Boston	0	10		Denver	Boston	10	8.3%
8		Denver	New York	50	0		Denver	New York	50	41.7%
9		Denver	Seattle	10	30		Denver	Seattle	40	33.3%

DATA | RESULTS

New ODBC Interface

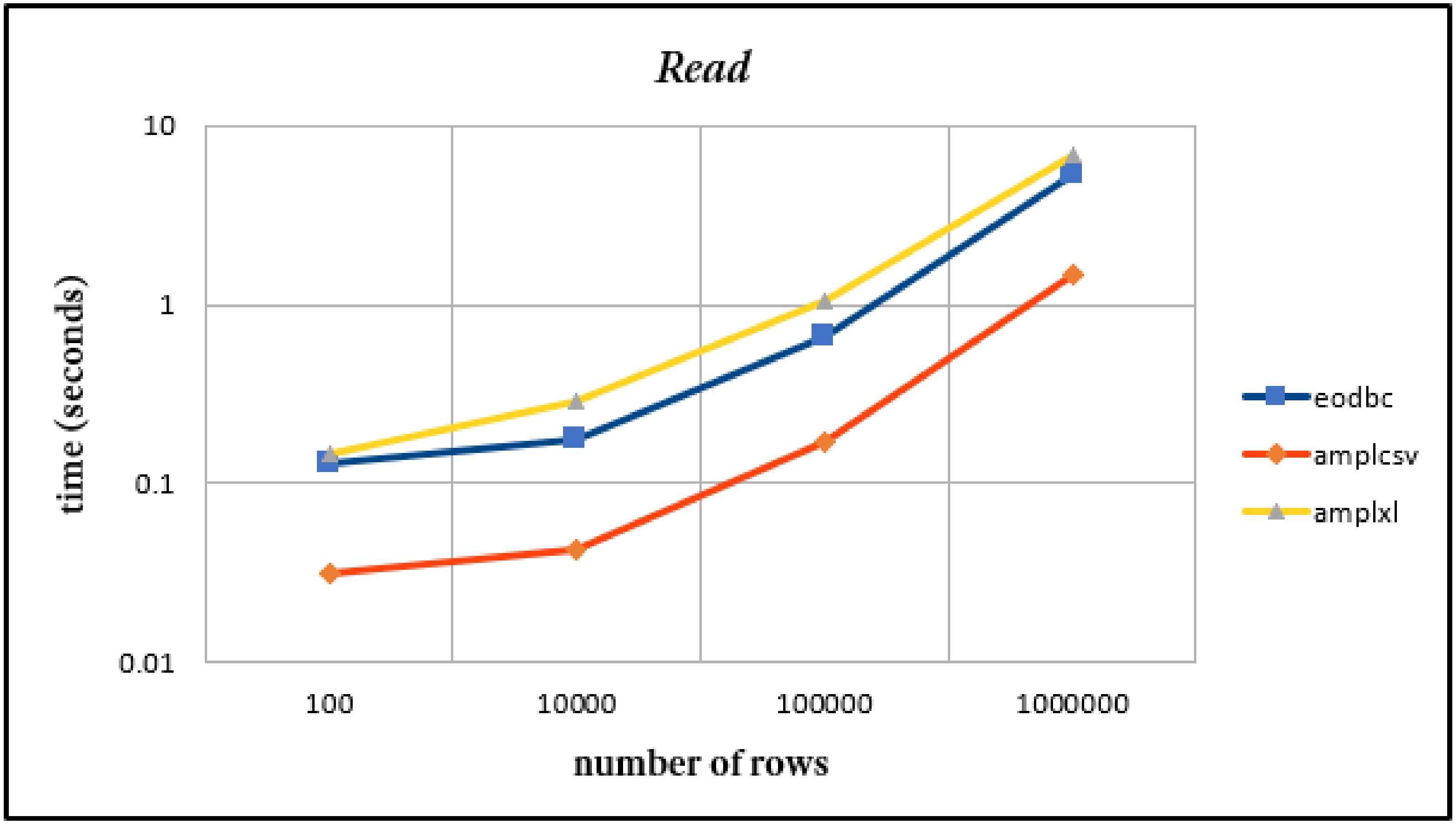
Open Database Connectivity

- Standard API for accessing database management systems
- Many database systems have ODBC drivers
- Supported by AMPL table statements
  - Can include SQL queries

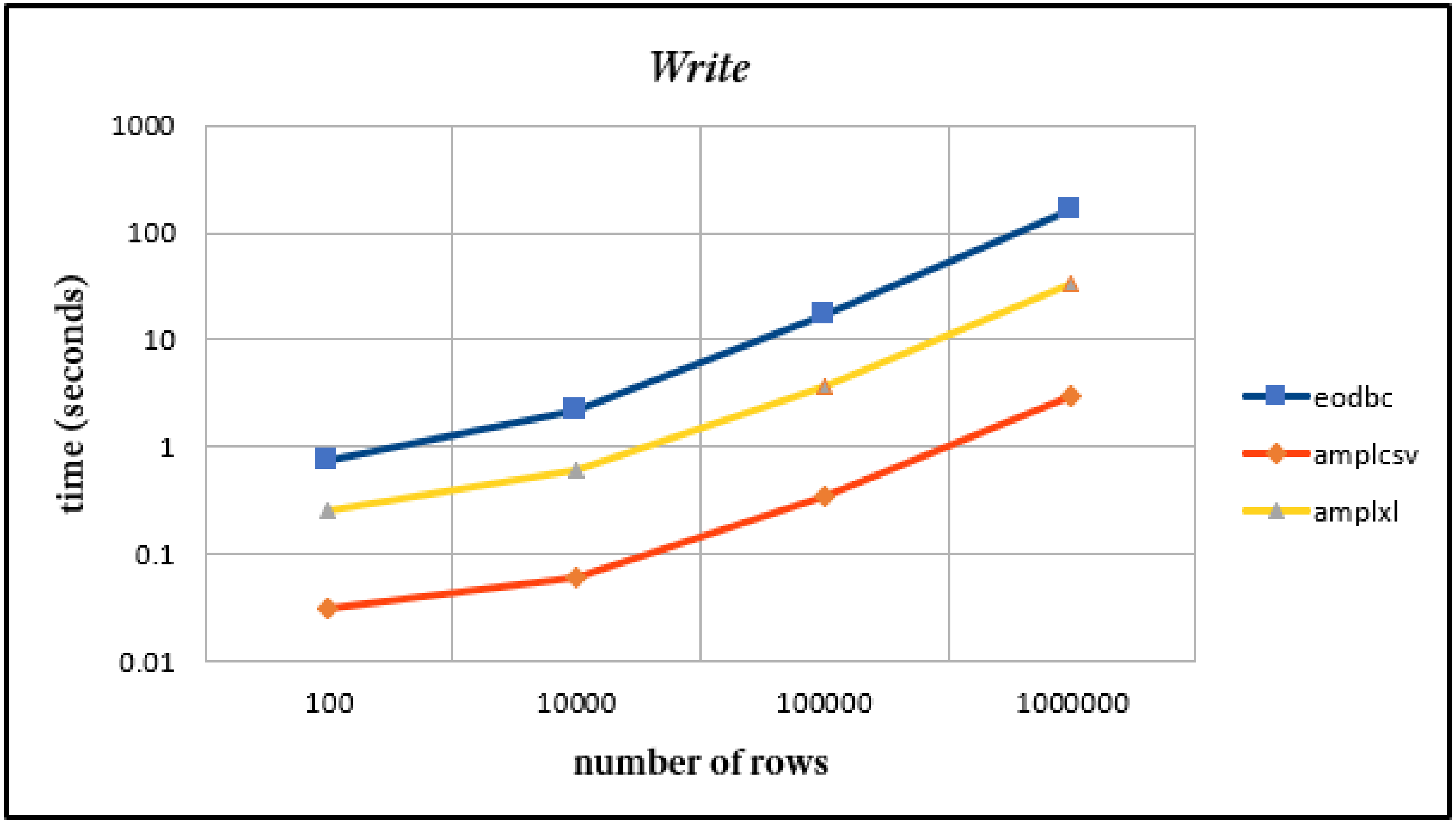
Enhancements

- Faster writes
- Table rewrite support
  - Preserve the column data types
- Table update support
  - Modify only selected records of a large table
- Table “upsert” support (experimental)
  - Update a record if it already exists, otherwise insert it
  - Requires a database-specific SQL statement

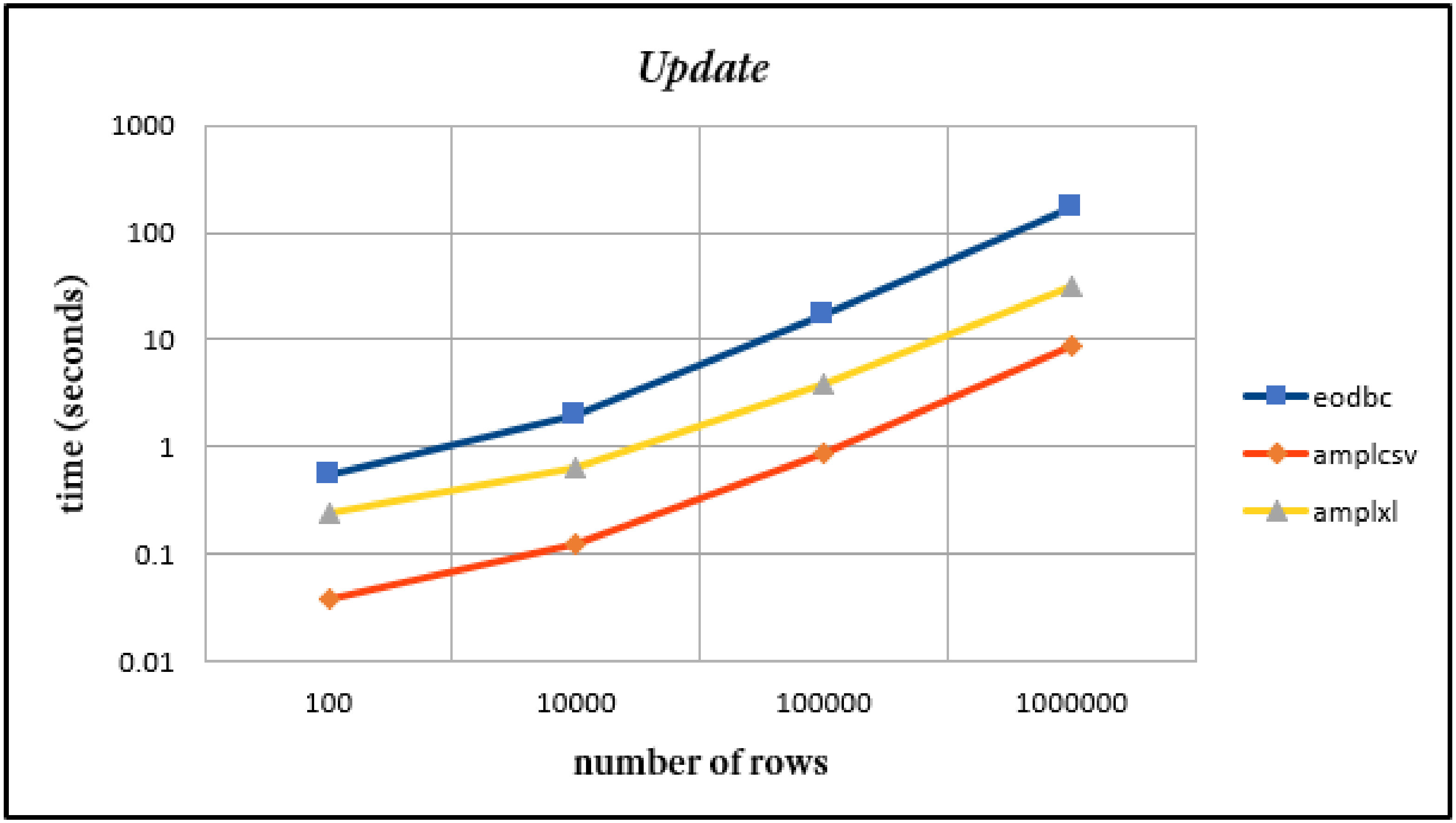
Reading Efficiency



Writing Efficiency



Updating Efficiency



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*

AMPL Environments

Native

- Interactive command line
- Model, data, and script (“run”) files

IDEs

- AMPL IDE, VScode

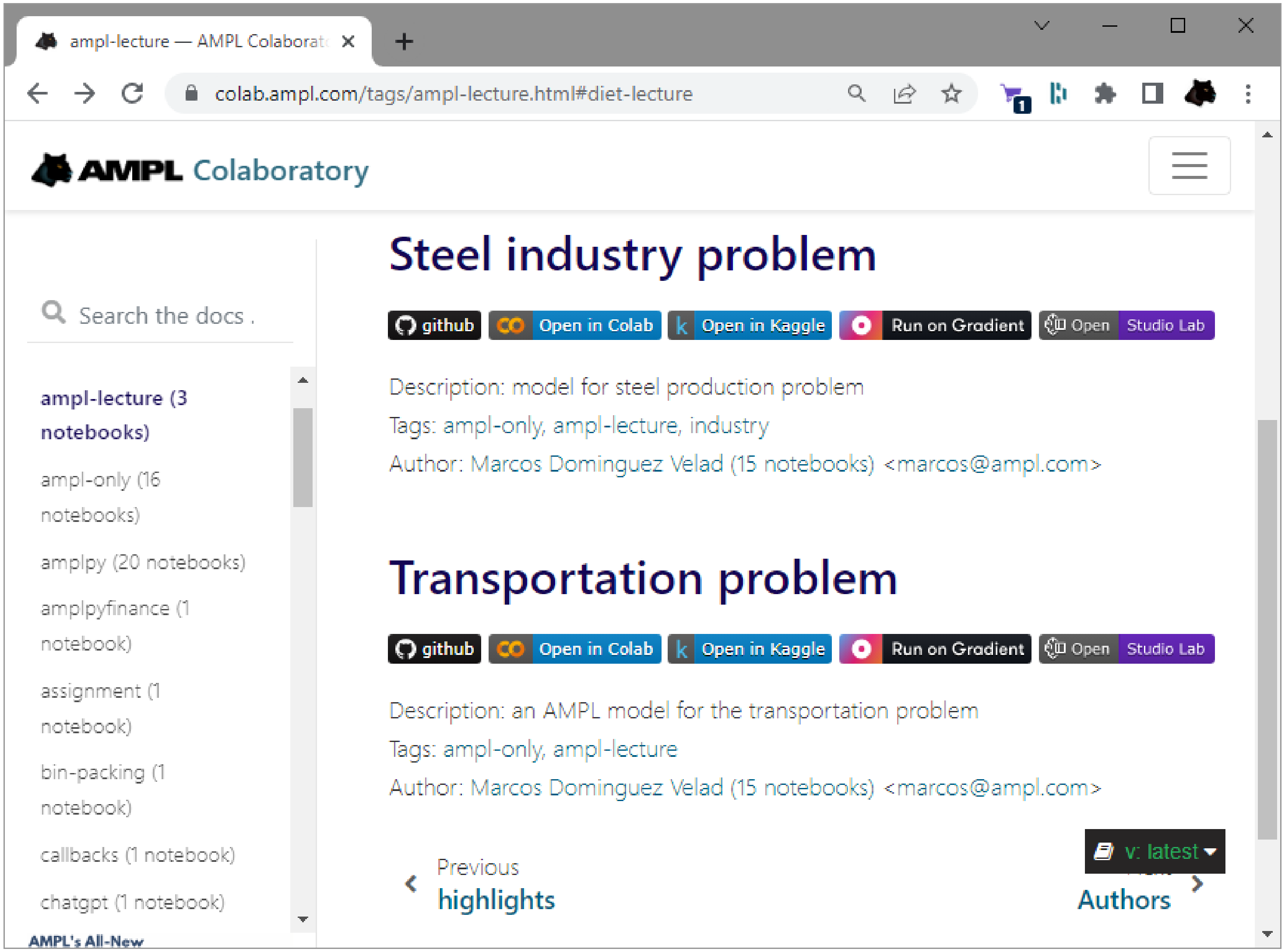
APIs

- C++, C#, Java, MATLAB, Python, R

Python

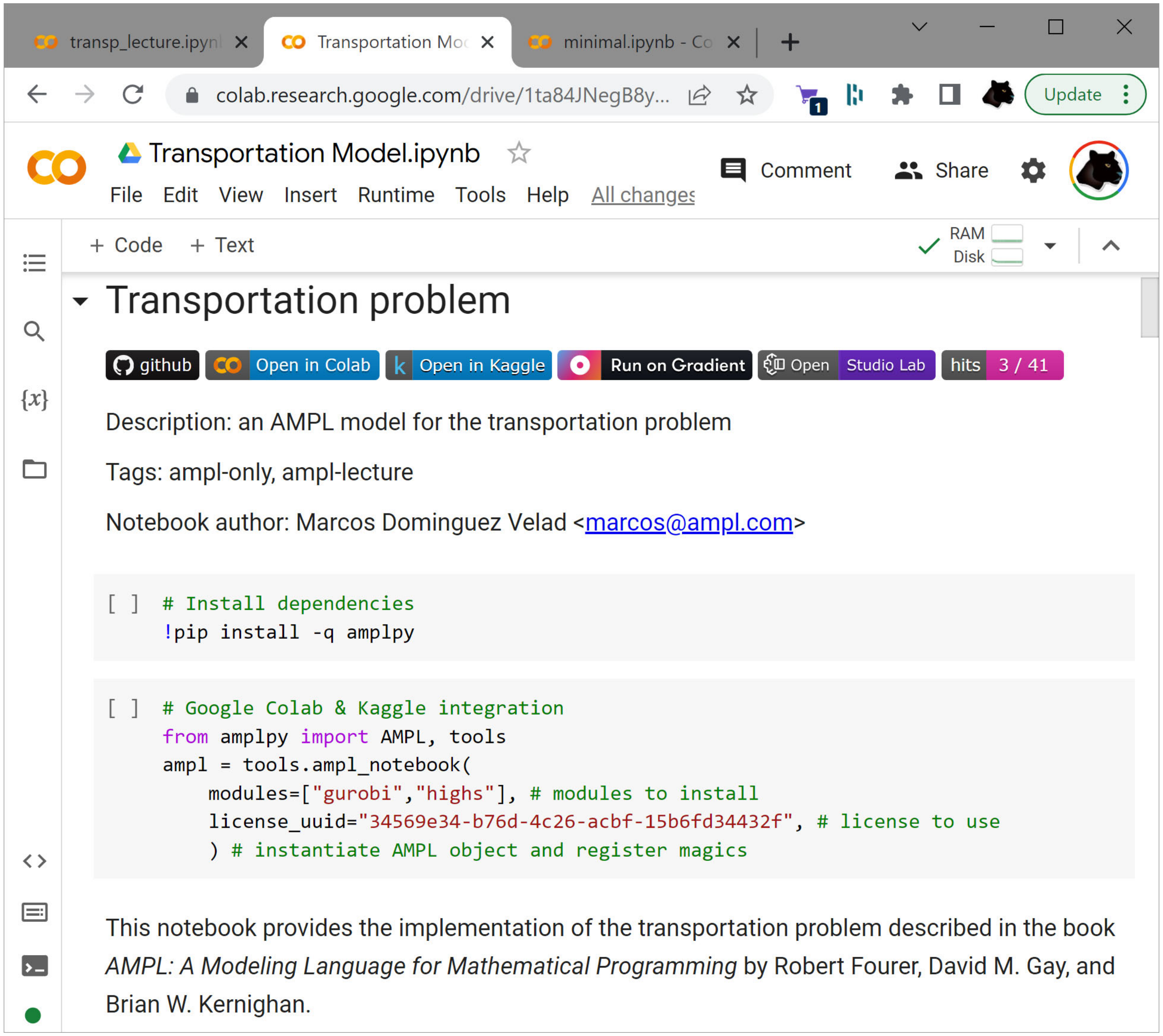
- Jupyter notebooks
- AMPL model colaboratory . . .

AMPL Model Colaboratory



AMPL Colaboratory

Opened in Google Colab



AMPL Colaboratory

Quick Setup

```
[1] # Install dependencies
    !pip install -q amplpy
```

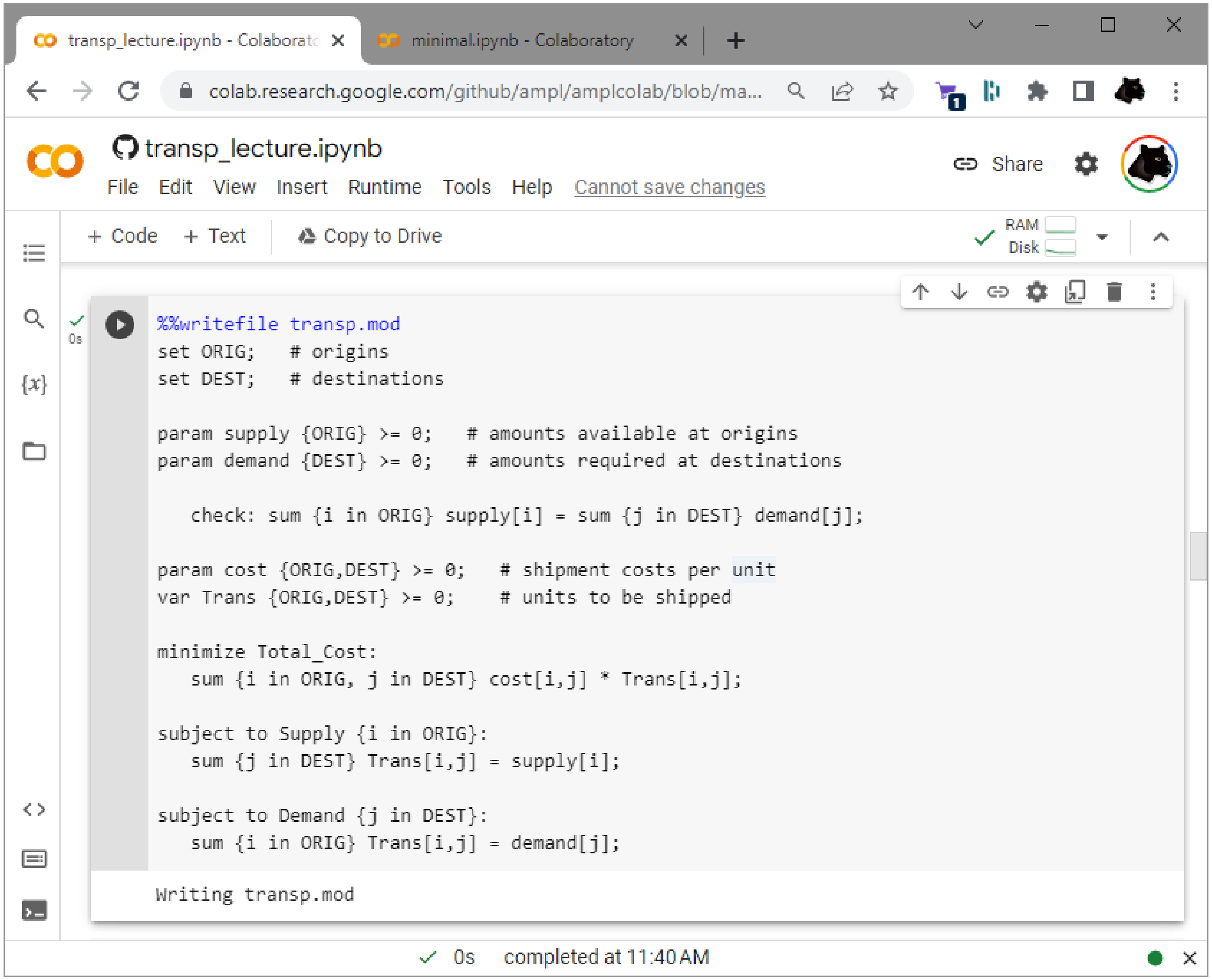
```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.5/4.5 MB 38.1 MB/s eta 0:00:00
```

```
# Google Colab & Kaggle integration
from amplpy import AMPL, tools
ampl = tools.ampl_notebook(
    modules=["gurobi","highs"], # modules to install
    license_uuid="34569e34-b76d-4c26-acbf-15b6fd34432f", # license to use
    ) # instantiate AMPL object and register magics
```

```
Licensed to Bundle #5883.6126 expiring 20230831: Test of course licenses, AMPL Optimi
```

AMPL Colaboratory

AMPL Model in Notebook Cell



AMPL Solve in Notebook Cell