Model-Based Optimization for Effective and Reliable Decision-Making

Robert Fourer

4er@ampl.com

AMPL Optimization Inc.

www.ampl.com — +1 773-336-2675

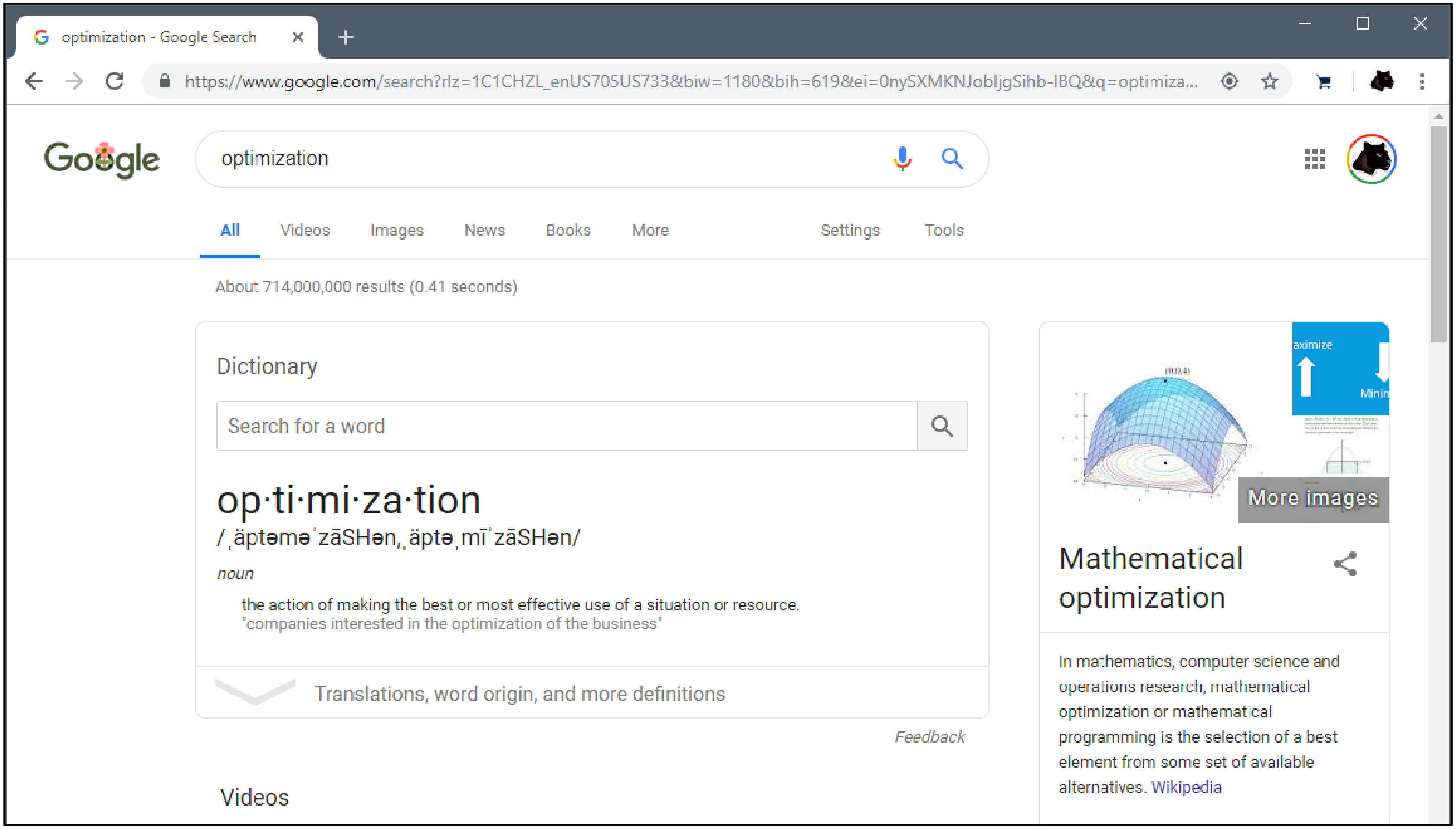
DecisionCAMP

Bolzano, Italy — 18 September 2019

Model-Based Optimization for Effective and Reliable Decision-Making

Optimization originated as an advanced mathematical technique, but it has become an accessible and widely used decision-making tool. A key factor in the spread of successful optimization applications has been the adoption of a model-based approach: A domain expert or operations analyst focuses on modeling the problem of interest, while the computation of a solution is left to general-purpose, off-the-shelf solvers; powerful yet intuitive modeling software manages the difficulties of translating between the human modeler's formulation and the solver software's needs. This talk introduces *model-based optimization* by contrasting it to a method-based approach that relies on customized implementation of rules and algorithms. Model-based implementations are illustrated using the AMPL modeling language and popular solvers. The presentation concludes by surveying the variety of modeling languages and solvers available for model-based optimization today.

Dr. Fourer has over 40 years' experience in studying, creating, and applying large-scale optimization software. In collaboration with colleagues in Computing Science Research at Bell Laboratories, he initiated the design and development of AMPL, which has become one of the most widely used software systems for modeling and analyzing optimization problems, with users in hundreds of universities, research institutes, and corporations worldwide; he is also author of a popular book on AMPL. Additionally, he has been a key contributor to the NEOS Server project and other efforts to make optimization services available over the Internet, and has supported development of open-source software for operations research through his service on the board of the COIN-OR Foundation.


optimization - Google Search
https://www.google.com/search?rlz=1C1CHZL_enUS705US733&biw=1180&bih=619&ei=0nySXMKNJobljgSihb-IBQ&q=optimiza...
optimization
All
Videos
Images
News
Books
More
Settings
Tools
About 714,000,000 results (0.41 seconds)
Dictionary
Search for a word
op·ti·mi·za·tion
/ˌäptəməˈzāSHən,ˌäptəˌmīˈzāSHən/
noun
the action of making the best or most effective use of a situation or resource.
"companies interested in the optimization of the business"
Translations, word origin, and more definitions
Feedback
Videos
More images
Mathematical optimization
In mathematics, computer science and operations research, mathematical optimization or mathematical programming is the selection of a best element from some set of available alternatives. Wikipedia

Mathematical optimization - Wik

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

In other projects

Wikimedia Commons

Print/export

Create a book
Download as PDF
Printable version

Languages

Deutsch
Español

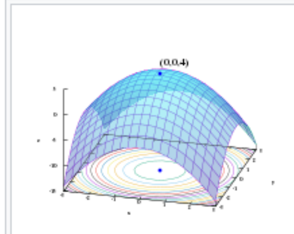
Article Talk Read Edit View history Search Wikipedia

Mathematical optimization

From Wikipedia, the free encyclopedia

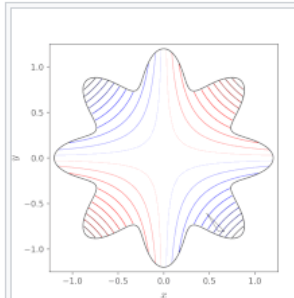
"Mathematical programming" redirects here. For the peer-reviewed journal, see Mathematical Programming.
"Optimization" and "Optimum" redirect here. For other uses, see Optimization (disambiguation) and Optimum (disambiguation).

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] Optimization problems of sorts arise in all quantitative disciplines from computer science and engineering to operations research and economics, and the development of solution methods has been of interest in mathematics for centuries.[2]



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.

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.



Nelder-Mead minimum search of Simionescu's function. Simplex vertices are ordered by their value, with 1 having the lowest (best) value.

Contents [hide]

1 Optimization problems
2 Notation
2.1 Minimum and maximum value of a function
2.2 Optimal input arguments
3 History
4 Major subfields
4.1 Multi-objective optimization
4.2 Multi-modal optimization
5 Classification of critical points and extrema
5.1 Feasibility problem
5.2 Existence
5.3 Necessary conditions for optimality
5.4 Sufficient conditions for optimality
5.5 Sensitivity and continuity of optima
5.6 Calculus of optimization
6 Computational optimization techniques

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] Optimization problems of sorts arise in all quantitative disciplines from computer science and engineering to operations research and economics, and the development of solution methods has been of interest in mathematics for centuries.[2]

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.

Optimization in Practice

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 *(formulations)*
- Ways of solving problems *(algorithms)*

Optimization in Practice

Large numbers of decision variables

- *Thousands to millions*

An objective function

Various constraint types

- *10-20 distinct types, though large numbers of each type*
- *Few variables involved in each constraint*

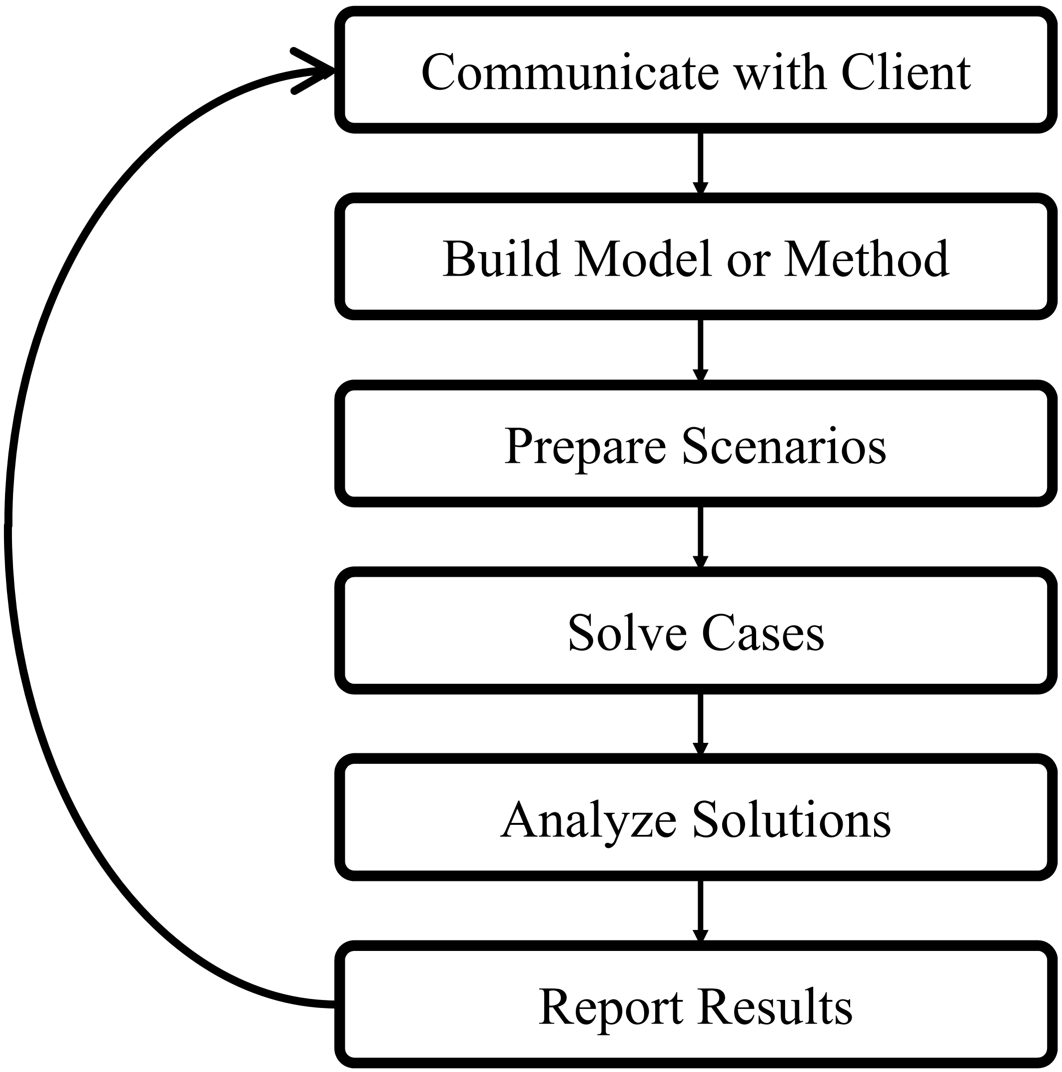
Solved many times with different data

- *Can't characterize all possible solutions in advance*

Solvable only by computation

- *No manual approaches even in principle*

The Optimization Modeling Cycle



The Optimization Modeling Cycle

Goals for the 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 (with decision systems)
- Successful long-term maintenance

Outline

Optimization

- The optimization modeling cycle
- Model-based vs. method-based approaches

Model-based optimization

- Modeling language vs. programming language approaches

Algebraic modeling languages

- Declarative vs. executable approaches
- Completed example in AMPL

Solvers

- Linear/quadratic, nonlinear, global, constraint-based

Applications

- Range of AMPL users
- Case studies

Where is the Work in Optimization?

It depends on the approach that you take

Method-based approach

- Programming a method (algorithm) for computing solutions

Model-based approach

- Formulating a description (model) of the desired solutions

Which should you prefer?

- For simple problems, any approach can be easy
- *But real optimization problems have complications . . .*

Example:

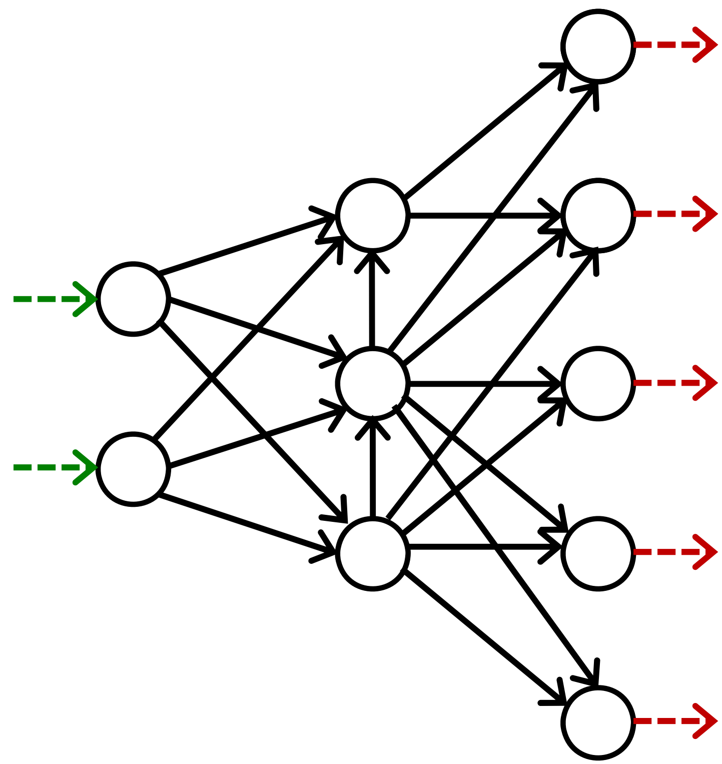
Multi-Product Optimal Network Flow

Motivation

- Ship products efficiently
  to meet demands

Context

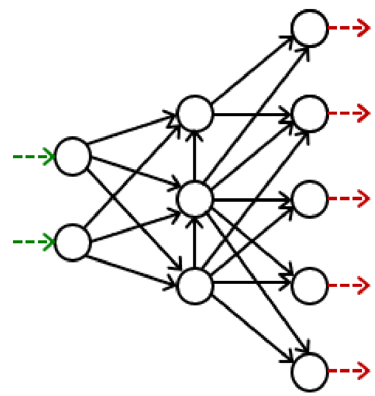
- a transportation network
  - nodes representing cities
  - arcs representing roads
- supplies at nodes
- demands at nodes
- capacities on arcs
- shipping costs on arcs



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

Two approaches . . .

Multi-Product Flow

Method-Based Approach

Program a method to build a shipping plan

- “method”: says how to compute a solution

Order-driven

- 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

- 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

Program refinements to the method to get better results . . .

Multi-Product Flow

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

- Handles broad problem classes efficiently
  - Ex: Linear constraints and objective, continuous or integer variables
- Recognizes and exploits special cases
- Available ready to run, without programming

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

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

on each arc, total shipped must not exceed capacity

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

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

Example:

Complications in Multi-Product Flow

Additional restrictions imposed by the user

- 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

$\sum_{p \in P} X_{pij} \leq u_{ij} Y_{ij},$ 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

$\sum_{p \in P} X_{pij} \geq m Y_{ij},$ 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

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

At most n arcs can be used

Method-Based Remains Popular for . . .

Applications of heuristic methods

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

Situations hard to formulate mathematically

- Difficult combinatorial constraints
- Black-box objectives and constraints

Very large, intensive applications

- Routing huge fleets of delivery trucks
- Finding shortest routes in mapping apps
- Training huge neural networks

. . . and it appeals to programmers

Model-Based Has Been Adopted in . . .

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 Has Been Adopted in . . .

Diverse industries

Diverse fields

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

Model-Based Has Been Adopted by . . .

Diverse industries

Diverse fields

Diverse kinds of users

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

These have in common . . .

- ❖ Analysts inclined toward modeling; focus is
  - ∗ more on *what* should be solved
  - ∗ less on *how* it should be solved
- ❖ Good algebraic formulations for off-the-shelf solvers
- ❖ Emphasis on fast prototyping *and* long-term maintenance

Where is the Work in Model-Based Optimization?

Translating 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
- Make formulations clearer and hence easier to steal?

Great advantages

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

Algebraic Modeling Languages

Most popular today

- Computer language based on *algebraic* formulations
  as seen in our model-based examples

Executable approach

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

Declarative approach

- Design a language specifically for optimization modeling
- Extend with basic programming concepts: loops, tests, assignments
- Access from popular programming languages via APIs

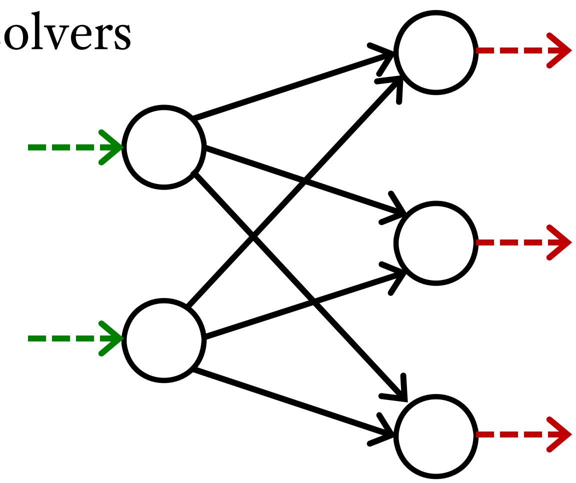
Example:
Multi-Product Optimal Network Flow

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 $p \in P$, $(i,j) \in A$

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 })
```

AMPL

- Define symbolic model sets and parameters

```
set PRODUCTS;
set NODES;

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

- Provide data later in a separate file →

```
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  ;
```

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")
```

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;

option solver gurobi;
ampl: solve;

Gurobi 8.1.0: optimal solution; objective 5500
2 simplex iterations

ampl: display Flow;

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

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

Multi-Product Flow

Solution *(cont'd)*

AMPL

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

option solver cplex;
ampl: solve;

CPLEX 12.9.0.0: optimal solution; objective 5500
0 dual simplex iterations (0 in phase I)

ampl: display Flow;

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

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

Integration with Applications

gurobipy

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

AMPL

- Modeling language extended with loops, tests, assignments
- Application programming interfaces (APIs) for calling AMPL from C++, C#, Java, MATLAB, Python, R
  - Efficient methods for data interchange

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

Complications

Easily accommodated

- Add variables to the model
- Add a term to the objective
- Update one constraint and add two
- Send to the same solver

See live example . . .

Algebraic Modeling Language Software

Solver-specific

- Associated with popular commercial solvers
  - IBM CPLEX, Gurobi, FICO Xpress
- Executable and declarative alternatives

Solver-independent

- Support multiple solvers and solver types
- Commercial options are mainly declarative
  - AIMMS, AMPL, GAMS
  - include APIs for popular programming languages
- Open-source options are mainly executable
  - CVX/MATLAB, FLOPC++/C++, JuMP/Julia, Pyomo/Python, YALMIP/MATLAB,

Solver Software

Off-the-shelf solvers for broad problem classes

- Based on optimal algorithms
- Implemented as complex methods + heuristics
- Adapted to special cases

Survey

Solver Software

Off-the-shelf solvers for broad problem classes

Many difficult problems solved regularly

- Millions of variables and constraints
- Hard problems of 10-20 years ago are now easy

Solver Software

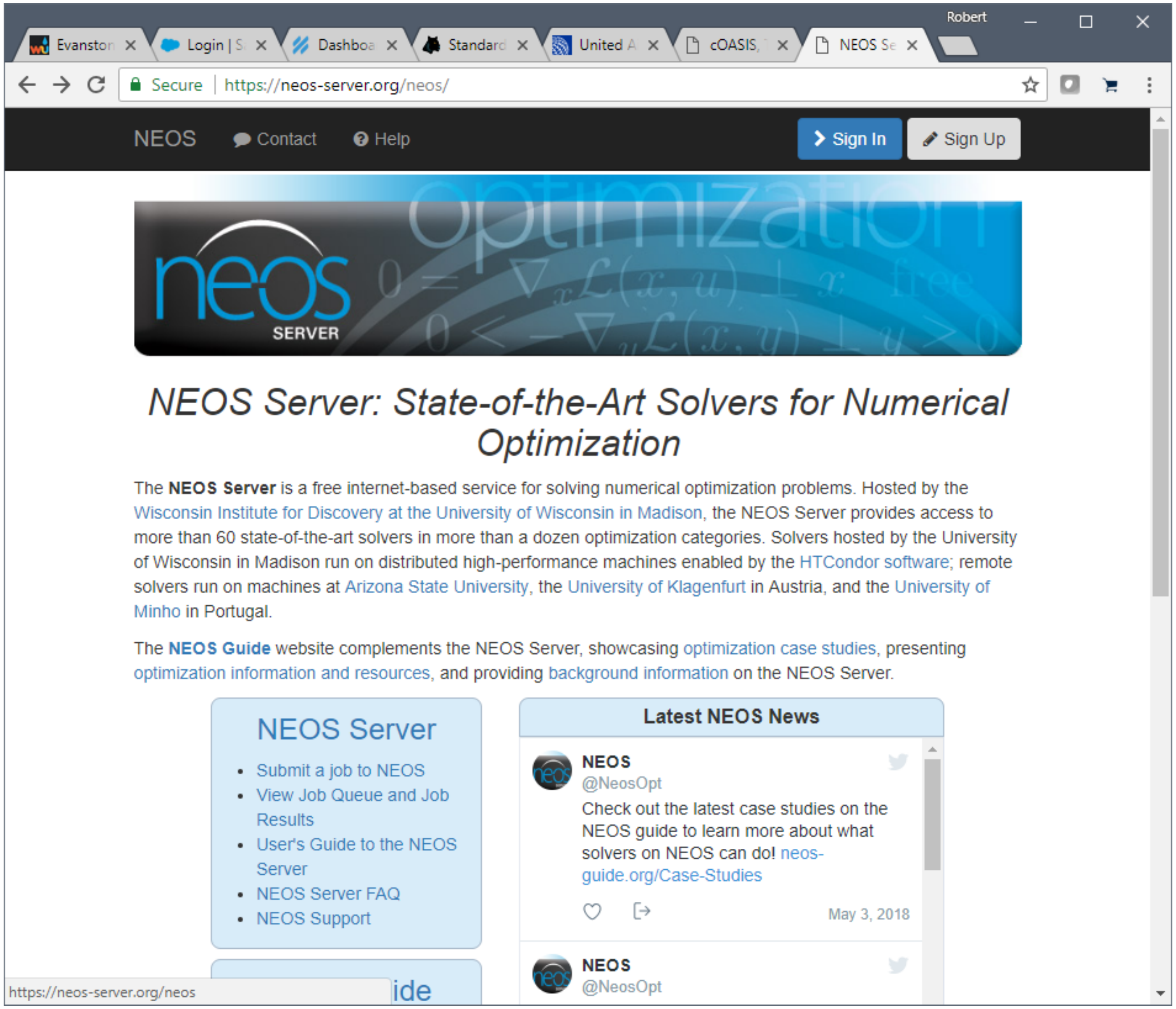
Off-the-shelf solvers for broad problem classes

Many difficult problems solved regularly

Commercial + open source examples

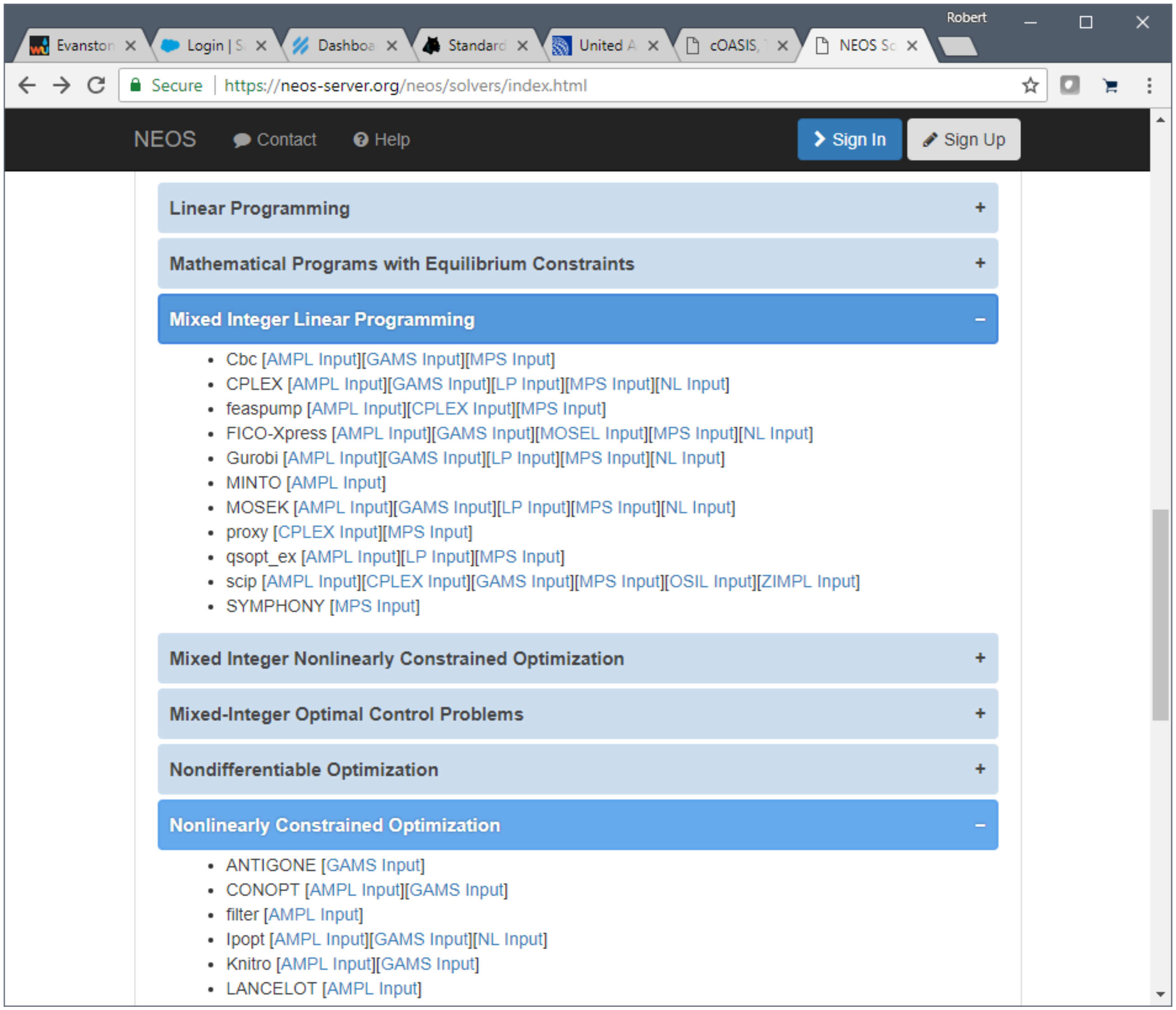
- "Linear/Quadratic":
  CPLEX, Gurobi, Xpress, MOSEK + SCIP, CBC, MIPCL
- "Nonlinear":
  CONOPT, Knitro, LOQO, MINOS, SNOPT + Ipopt, Bonmin
- "Global":
  BARON, LINDO Global + Couenne
- "Constraint":
  IBM ILOG CP + Gecode, JaCoP

Curious? Try Them Out on NEOS!



NEOS Server

Solver & Language Listing



Applications

Range of AMPL users

Case studies

- Power grid management
- Passenger flow management
- Sales representative assignment

Range of AMPL Users

Energy and Utilities

- power networks, gas pipelines, hydroelectric power, water distribution

Industry

- mining, steel, chemicals, oil refining, forestry and paper
- cars & trucks, paper products, processed foods

Transportation

- airlines, trucking, package delivery

Services

- supply chain, hospitals & medicine, construction management

Communications

- telecommunications, professional networking, file hosting

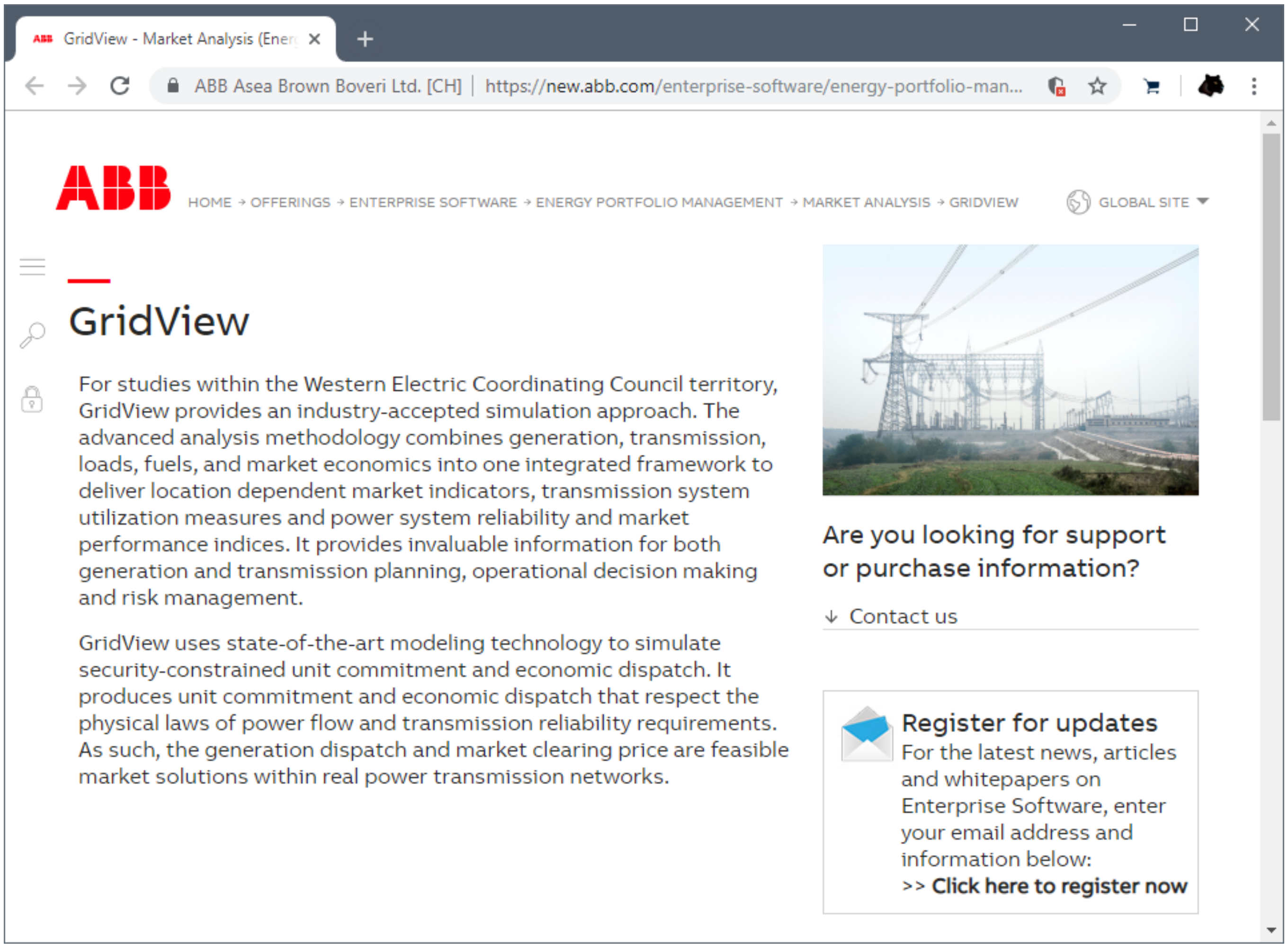
Finance

- software tools, investment management, commodity management

Advanced Technologies

- artificial intelligence, distributed computing, biotechnology

Case: ABB
Power Grid Management



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 Management

Situation

- A power grid operator providing 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

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

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 customize models for their particular situations
- *Ease of embedding*
  - AMPL has an API (application programming interface) for C++
  - ABB can easily build AMPL into the GridView product

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

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

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 3 analysts at ABB

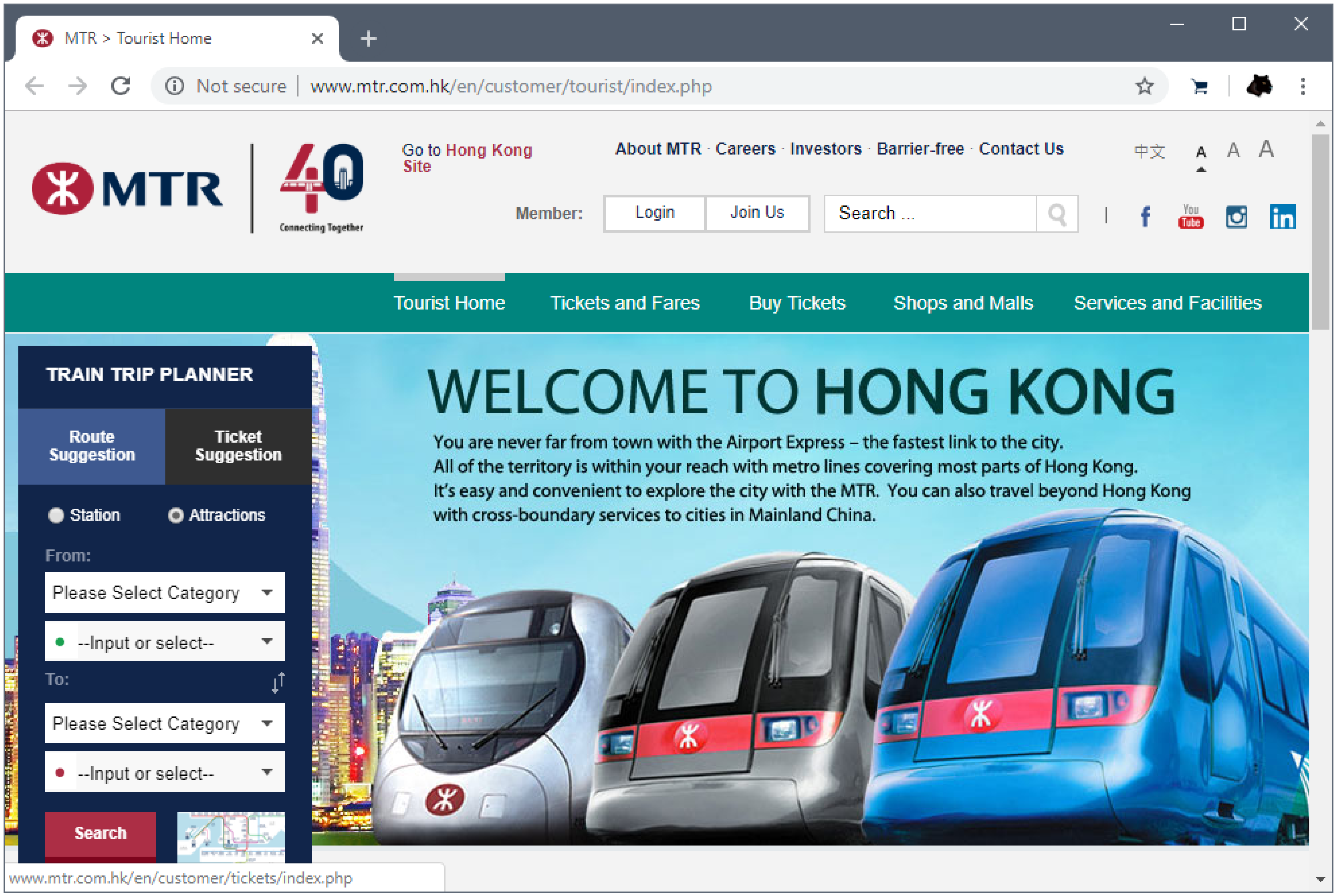
Optimization

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

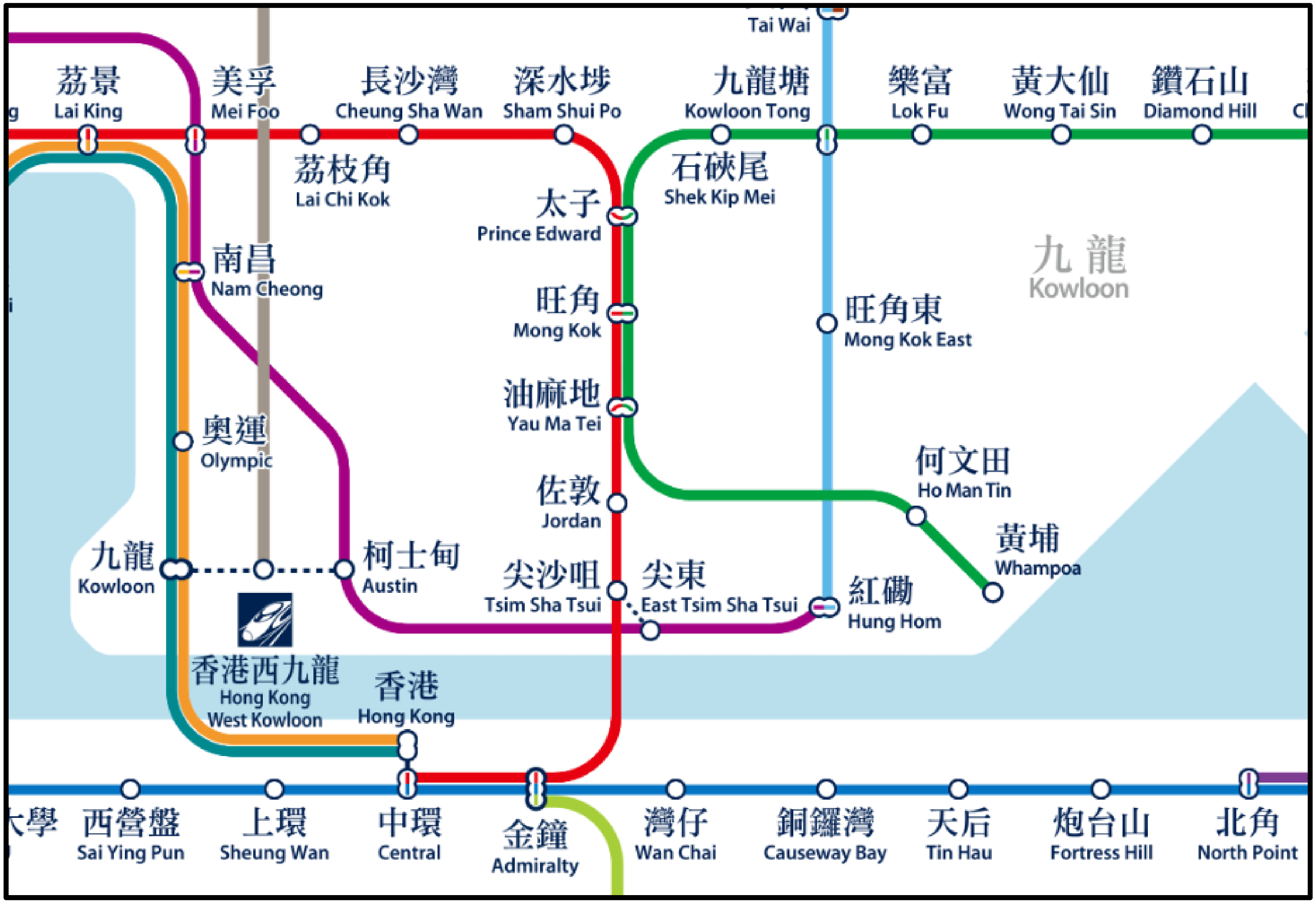
Deployment

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

Case: MTR HK / Strategis Partners
Passenger Flow Management



Passenger Flow Management



Passenger Flow Management

Situation

- Large public train operator
- 2 million passengers in 2 hours each weekday afternoon
- Arrivals exceed capacity
  - More passengers arrive on a platform than a train can handle
- Measures are in place that can limit entry to station & platforms

Goal

- Decide where and when to implement passenger-limiting measures
- Balance platform use throughout the system

Evaluation

Approaches

- *Old:* Best guesses of experience managers
- *New:* Modeling language for optimization,
  R to manipulate input data and display results

Choice of AMPL

- *Ease of use*
  - Convenient model syntax
  - Speed of processing
- *Ease of embedding*
  - AMPL is easily built into an R application,
    using AMPL's API (application programming interface) for R

Formulation *(data and variables)*

Data

- Design of the train network
- Passenger entry and intended exit stations
  - Supplied by the ticketing system
- Platform capacities

Decision variables

- For each time interval, at each station, for each train service:
  - How many passengers to allow in to the platform
  - How many passengers to expect out at the platform

Formulation *(objective and constraints)*

Objective

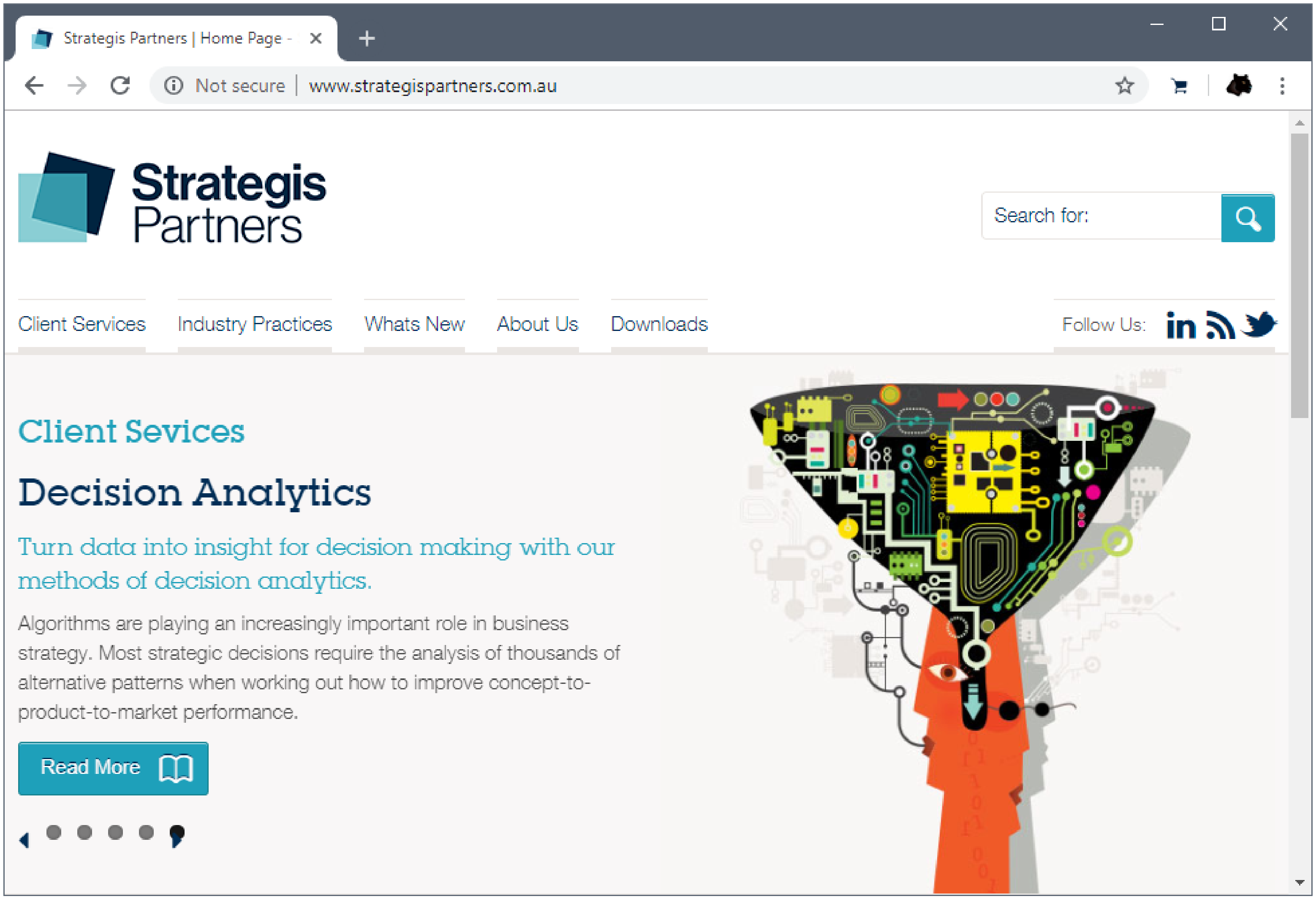
- Minimize aggregate passenger travel times across the network

Constraints

- Train travel times
- Train capacities
- Station concourse capacities

Passenger Flow Management

Implementation



Implementation

Development

- Strategis Partners consulting firm
- AMPL and R implementation by 2 analysts

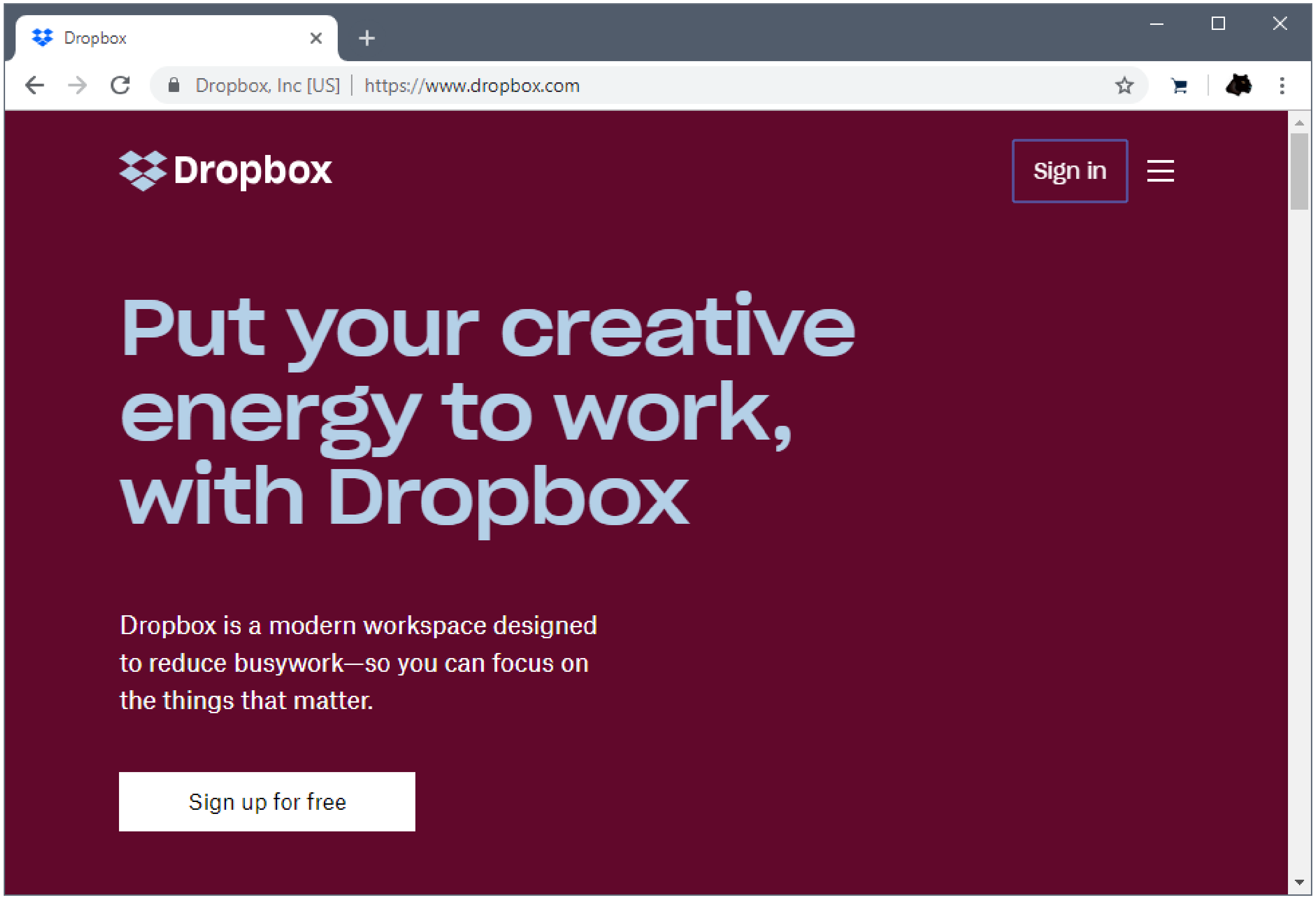
Optimization

- Free open-source mixed-integer linear solver
- 15 core stations
- 250,000 variables and constraints
- 20 minutes to solve

Deployment

- 2 users at MTR HK run as needed
- Extensions and enhancements planned
  - using machine learning to forecast passenger flows
  - expanding to more stations

Case: Dropbox
Sales Representative Assignment



Sales Representative Assignment

Situation

- Cloud storage provider
- Over 500 million users upload 1.2 billion files every day
- Tens of thousands of large business customer accounts
- Hundreds of sales representatives worldwide
  - enough to cover most but not all accounts

Goal

- Assign accounts to representatives
  - Assign each representative a similar number and quality of accounts
  - Give priority to assigning higher quality accounts

Evaluation

Approaches considered

- Manual system
- Spreadsheet-based solvers
- Automated system using model-based optimization

Choice of AMPL

- Ease of use
- Speed
- Reliability
- Ability to handle large problems

Formulation *(data and variables)*

Data

- Quality *score* for each customer account
  - predicted revenue increase if contacted by a representative
- Location of each representative

Decision variables

- For each account i and representative j,
  $X_{ij} = 1$ if account i is assigned to representative j
  $X_{ij} = 0$ otherwise

Formulation *(objective and constraints)*

Objective

- Maximize total score of all assigned accounts

Constraints

- At most 15% variance between representatives in . . .
  - number of accounts assigned
  - quality of accounts assigned
- Assigned accounts must be near the representative's location
- All subaccounts of a business must have the same representative

Sales Rep Assignment

Implementation

Development

- ❖ Implementation by 3 analysts at Dropbox

Optimization

- ❖ Mixed-integer linear solver
- ❖ 10,000 zero-one variables
- ❖ 3-6 hours to solve for largest region

Deployment

- ❖ 5-10 sales leaders are direct users
- ❖ AMPL is embedded in Dropbox's systems
  - ∗ Customer data is extracted from *Salesforce*
  - ∗ Customer scores are computed using the *scikit-learn* Python toolbox
  - ∗ An AMPL script reads the file of score data
  - ∗ Results from optimization are written to an *Excel* spreadsheet