Model-Based Optimization

PLAIN AND SIMPLE

From Formulation to Deployment with AMPL

AMPL Optimization Inc.

www.ampl.com — +1 773-336-AMPL

INFORMS Business Analytics Conference

Technology Workshop, 15 April 2018

Model-Based Optimization, Plain and Simple: From Formulation to Deployment with AMPL

Optimization is the most widely adopted technology of Prescriptive 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 integrate optimization into your enterprise's decision-making systems?
- How can you deploy optimization models to support analysis and action throughout your organization?

In this presentation, we show how AMPL gets you going without elaborate training, extra programmers, or premature commitments. 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 while maintaining ease of use.

The remainder of the presentation takes a single example through successive stages of the optimization modeling lifecycle:

- Prototyping in an interactive command environment.
- Integration via AMPL scripts and through APIs to all popular programming languages.
- Deployment with QuanDec, which turns an AMPL model into an interactive, collaborative decision-making tool.

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.

Outline

Part 1. Model-based optimization, plain and simple

`https://ampl.com/MEETINGS/TALKS/2018_04_Baltimore_Workshop1.pdf`

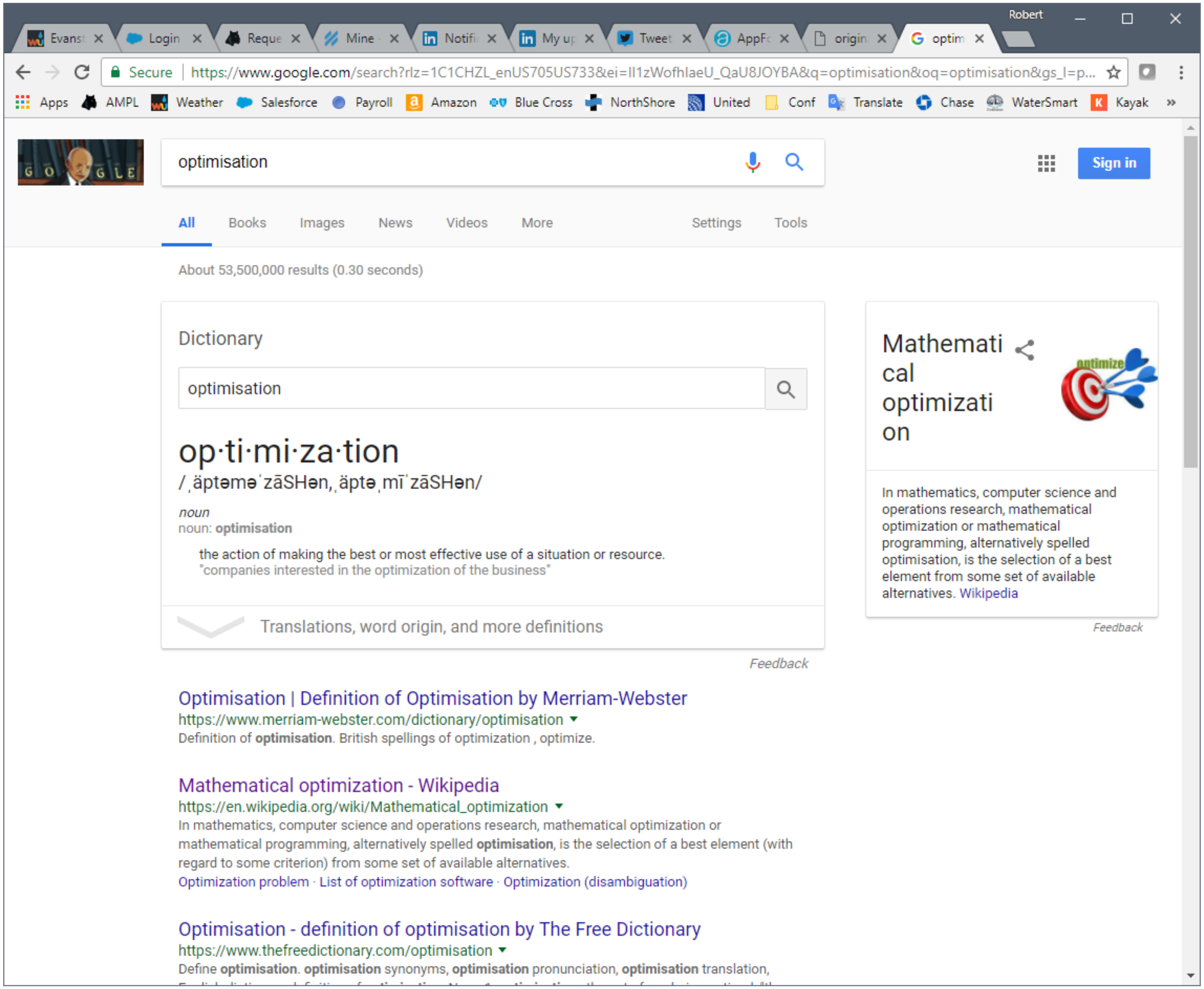
- Comparison of *method-based* and *model-based* approaches
- Modeling languages for optimization
- Algebraic modeling languages: AMPL
- Solvers for broad model classes

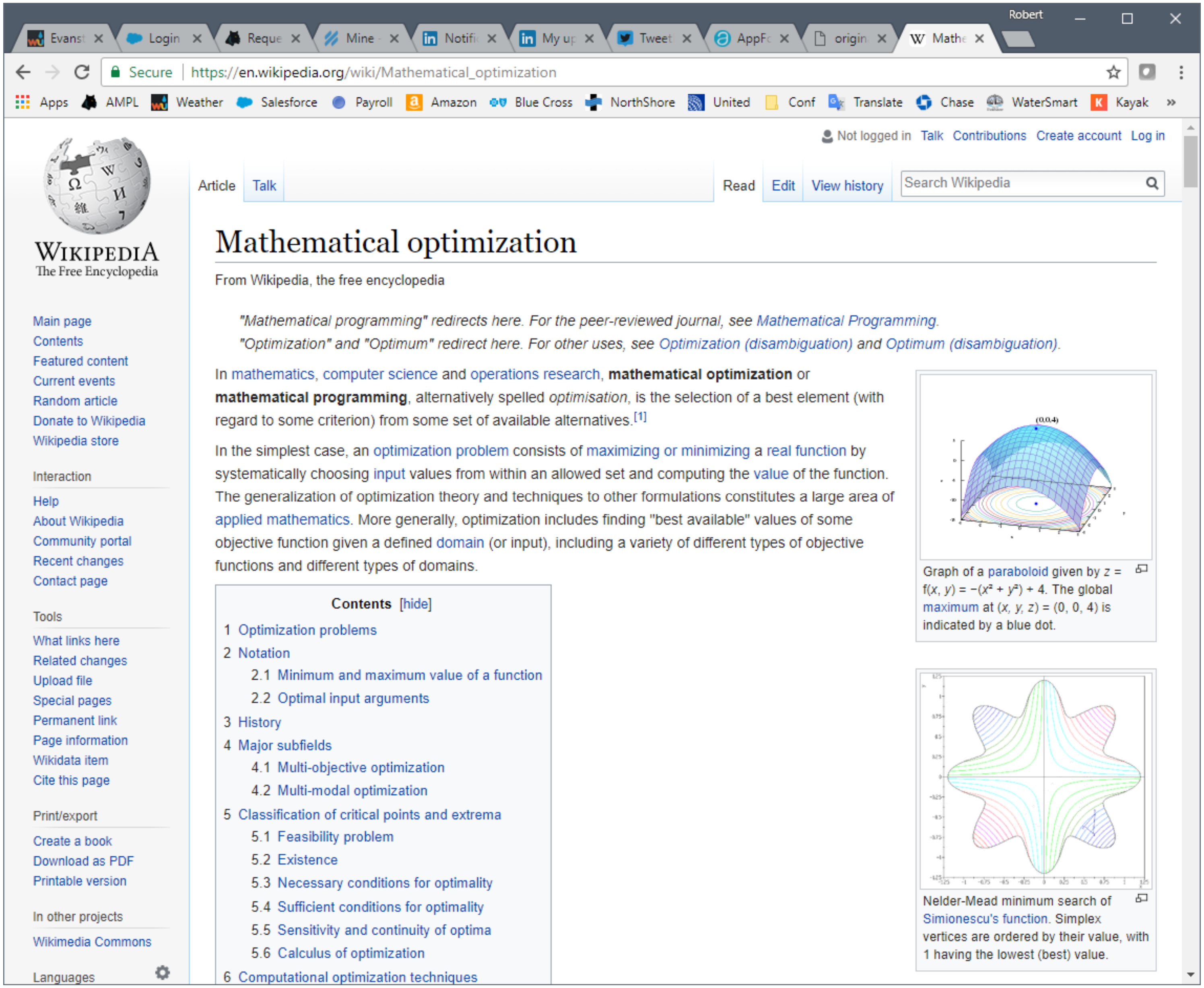
Part 2. From formulation to deployment with AMPL

`https://ampl.com/MEETINGS/TALKS/2018_04_Baltimore_Workshop2.pdf`

- Building models: *AMPL's interactive environment*
- Developing applications: *AMPL scripts*
  - Extending script applications with Python: *pyMPL*
- Embedding into applications: *AMPL APIs*
- Creating an interactive decision-making tool: *QuanDec*

Part 1
Model-Based Optimization, Plain and Simple


Robert
https://www.google.com/search?rlz=1C1CHZL_enUS705US733&ei=Il1zWofhlaeU_QaU8JOYBA&q=optimisation&oq=optimisation&gs_l=p...
Apps AMPL Weather Salesforce Payroll Amazon Blue Cross NorthShore United Conf Translate Chase WaterSmart Kayak
optimisation
Sign in
All Books Images News Videos More Settings Tools
About 53,500,000 results (0.30 seconds)
Dictionary
optimisation
op·ti·mi·za·tion
/ˌäptəməˈzāSHən,ˌäptəˌmīˈzāSHən/
noun
noun: optimisation
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
Optimisation | Definition of Optimisation by Merriam-Webster
https://www.merriam-webster.com/dictionary/optimisation
Definition of optimisation. British spellings of optimization , optimize.
Mathematical optimization - Wikipedia
https://en.wikipedia.org/wiki/Mathematical_optimization
In mathematics, computer science and operations research, mathematical optimization or mathematical programming, alternatively spelled optimisation, is the selection of a best element (with regard to some criterion) from some set of available alternatives.
Optimization problem · List of optimization software · Optimization (disambiguation)
Optimisation - definition of optimisation by The Free Dictionary
https://www.thefreedictionary.com/optimisation
Define optimisation. optimisation synonyms, optimisation pronunciation, optimisation translation,
Mathematical optimization
optimize
In mathematics, computer science and operations research, mathematical optimization or mathematical programming, alternatively spelled optimisation, is the selection of a best element from some set of available alternatives. Wikipedia
Feedback



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

Not logged in Talk Contributions Create account Log in

Article Talk Read Edit View history Search Wikipedia

WIKIPEDIA
The Free Encyclopedia

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

Interaction

Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools

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

Print/export

Create a book
Download as PDF
Printable version

In other projects

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** or **mathematical programming**, alternatively spelled *optimisation*, is the selection of a best element (with regard to some criterion) from some set of available alternatives.[1]

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

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

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

In general terms,

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

In practice,

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

Example: Balanced Assignment

Motivation

- meeting of employees from around the world

Given

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

Assign

- each employee to a project group

So that

- the groups have about the same size
- *the groups are as "diverse" as possible*

Method-Based Approach

Define an algorithm to build a balanced assignment

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

```
Initialize all groups G = { }

Repeat for each person p
   sMin = Infinity

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

      if s < sMin then
         sMin = s
         GMin = G

   Assign person p to group GMin
```

Method-Based Approach *(cont'd)*

Define a computable concept of "sameness"

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

Refine the algorithm to get better results

- ❖ Order the list of people
- ❖ Locally improve the initial "greedy" solution by swapping group members
- ❖ Seek further improvement through local search metaheuristics

Model-Based Approach

Formulate a "minimal sameness" model

- Define decision variables for assignment of people to groups
- Specify valid assignments through constraints on the variables
- Formulate sameness as an objective to be minimized
  - *Total sameness* = sum of the sameness of all groups

Send to an off-the-shelf solver

- Choice of many for common problem types

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

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

To minimize

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

total sameness of all pairs of people in all groups

Subject to

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

each person must be assigned to one group

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

each group must be assigned an acceptable number of people

Model-Based Solution

Optimize with an off-the-shelf solver

Choose among many alternatives

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

Where Is the Work?

Method-based

- Programming an implementation of the method

Model-based

- Constructing a formulation of the model

Complications in Balanced Assignment

"Total Sameness" is problematical

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

Client has special requirements

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

Room capacities are variable

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

Method-Based *(cont'd)*

Revise or replace the initial solution method

- ❖ Total variation is less suitable to a greedy algorithm

Re-think improvement procedures

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

Revise or re-implement the method

- ❖ Even small changes to the problem
can necessitate major changes to the method

Model-Based *(cont'd)*

Add variables

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

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

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

Add defining constraints

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

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

Minimize total variation

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

. . . generalizes to handle varying group sizes

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

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

Balanced Assignment

Model-Based *(cont'd)*

To express client requirement for women in a group, let

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

Define logic variables

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

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

Add constraints relating logic to assignment variables

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

Model-Based *(cont'd)*

To express client requirements for group-mates, let

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

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

Add constraints

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

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

Model-Based *(cont'd)*

To express client requirements for group-mates, let

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

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

Define logic variables

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

Add constraints relating logic to assignment variables

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

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

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

Method-Based Remains Attractive for . . .

Problems that are challenging to formulate

- Certain types of complex logic
  - Sequencing, scheduling
- Black-box functions

Very large, specialized problems embedded in apps

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

Metaheuristic frameworks

- Evolutionary methods, simulated annealing, . . .

Applications associated with computer science

- Constraint programming
- Training deep neural networks

Model-Based Has Become Standard for . . .

Diverse application areas

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

Diverse kinds of users

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

. . . and trends favor this direction

- Steadily faster and more powerful off-the-shelf solvers
- *Expanding options to incorporate models within hybrid schemes*

Software for Model-Based Optimization

Background

- The modeling lifecycle
- Matrix generators
- Modeling languages

Algebraic modeling languages

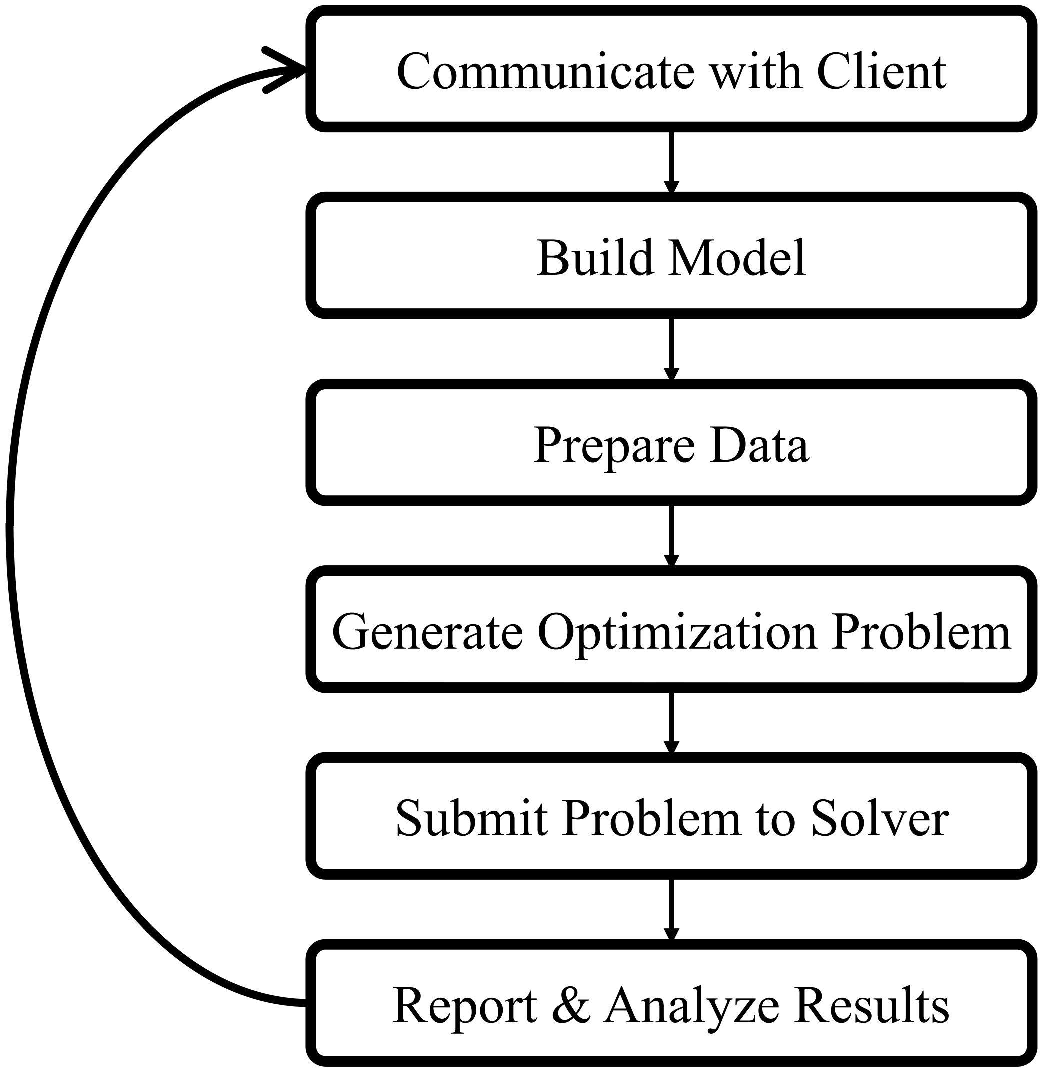
- Design & marketing approaches
- AMPL: General-purpose, solver-independent

Balanced assignment model in AMPL

- Formulation
- Solution

Solvers

The Optimization Modeling Lifecycle



Managing the Modeling Lifecycle

Goals for optimization software

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

Complication: two forms of an optimization problem

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

Challenge: translate between these two forms

Matrix Generators

Write a program

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

Some attractions

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

Serious disadvantages

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

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

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

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

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

Modeling Languages

Describe your model

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

Limited drawbacks

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

Great advantages

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

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

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

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

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

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

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

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

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

Algebraic Modeling Languages

Algebraic formulation

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

Advantages

- Familiar
- Powerful
- Proven

Algebraic Modeling Languages

Design approaches

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

Marketing approaches

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

Executable

Concept

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

Advantages

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

Disadvantages

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

Executable

Examples (Gurobi/Python)

```
model.addConstrs(x[i] + x[j] <= 1
   for i in range(5) for j in range(5))
```

```
for i,j in arcs:
   m.addConstr(gurobipy.quicksum(flow[h,i,j] for h in commodities)
      <= capacity[i,j], 'cap_%s_%s' % (i, j))
```

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.

Executable

Licensed, solver-specific

- C++: CPLEX
- Python; Gurobi, SAS
- MATLAB; Optimization Toolbox

Open-source, solver-independent

- Python: Pyomo, PuLP
- MATLAB: YALMIP, CVX
- Julia: JuMP
- C++: FLOPC++, Rehearse

Declarative

Concept

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

Disadvantages

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

Advantages

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

Declarative

Solver-specific

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

Solver-independent

- Open-source: CMPL, Gnu MathProg
- Licensed: AIMMS, ***AMPL,*** GAMS, MPL

Declarative

Many enhancements and extensions

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

AMPL

Features

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

Design goals

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

Modeling Language Formulation

Sets, parameters, variables (for people)

```
set PEOPLE;   # individuals to be assigned

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

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

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

              # number of groups; bounds on size of groups

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

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

Modeling Language Formulation

Variables, constraints (for variation)

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

              # all types found in each category

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

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

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

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

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

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

Modeling Language Formulation

Objective, constraints (for assignment)

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

              # Total variation over all types

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

              # Each person must be assigned to one group

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

              # Each group must have an acceptable size
```

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

Modeling Language Data

210 people

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

Modeling Language Data

4 categories, 18 types

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

param type:
     dept        loc      rate    title   :=

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

.......

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

Modeling Language Solution

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

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

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

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

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

8.063 sec

Modeling Language Solution

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

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

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

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

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

66.344 sec

Modeling Language Solution

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

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

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

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

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

61.125 sec

Modeling Language Formulation *(revised)*

Add bounds on variables

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

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

```
ampl: include BalAssign+.run

Presolve eliminates 72 constraints.
...

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

0.203 sec

Solvers: “Linear”

CPLEX, Gurobi, Xpress; CBC, MOSEK

Linear

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

Quadratic extensions

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

Solvers: “Nonlinear”

CONOPT, Knitro, LOQO, MINOS, SNOPT; Bonmin, Ipopt

Continuous variables

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

Extension to integer variables: Knitro, Bonmin

Automatic multistart: Knitro

Other Useful Solvers

BARON; Couenne

- Continuous and integer variables
- Smooth nonlinear objective and constraint functions
- Globally optimal solutions

LGO

- Continuous nonlinear objective and constraint functions not necessarily smooth or convex
- High-quality solutions, may be global

ILOG CP; Gecode, JaCoP

- Integer variables
- Logical conditions directly in constraints; encoding of logic in binary variables not necessary
- Globally optimal solutions