Model-Based Optimization

Principles and Trends

Robert Fourer
AMPL Optimization Inc.
4er@ampl.com

24th International Conference on
Principles and Practice of Constraint Programming — Lille, France
Tutorial 1, 28 August 2018, 14:00-15:00
Model-Based Optimization: Principles and Trends

As optimization methods have been applied more broadly and effectively, a key factor in their success has been the adoption of a model-based approach. A researcher or analyst focuses on modeling the problem of interest, while the computation of a solution is left to general-purpose, off-the-shelf solvers; independent modeling languages and systems manage the difficulties of translating between the human modeler’s ideas and the computer software’s needs. This tutorial introduces model-based optimization with examples from the AMPL modeling language and various popular solvers; the presentation concludes by surveying current software, with special attention to the role of constraint programming.
Outline

Approaches to optimization
  ❖ Model-based vs. Method-based

Modeling languages for model-based optimization
  ❖ Motivation for modeling languages
  ❖ Algebraic modeling languages
  ❖ Executable vs. declarative languages
  ❖ Survey of modeling language software

Solvers for model-based optimization
  ❖ Linear
  ❖ Nonlinear
  ❖ Global
  ❖ Constraint
Examples

**Approaches to optimization**
- Model-based vs. Method-based
  - Example: Balanced assignment

**Modeling languages for model-based optimization**
- Executable vs. declarative languages
  - Example: gurobipy vs. AMPL
- Survey of modeling language software
  - Example: Balanced assignment in AMPL
  - Example: Nonlinear optimization in AMPL

**Solvers for model-based optimization**
- Constraint
  - Example: Balanced assignment via CP in AMPL
Example: Balanced Assignment

Motivation
- meeting of employees from around the world

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

Assign
- each employee to a project group

So that
- the groups have about the same size
- the groups are as “diverse” as possible with respect to all categories
**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

```plaintext
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
```
**Balanced Assignment**

**Method-Based Approach (cont’d)**

**Define a computable concept of “sameness”**

- Sameness of any two people:
  - Number of categories in which they are the same
- Sameness of a group:
  - Sum of the sameness of all pairs of people in the group

**Refine the algorithm to get better results**

- Reorder the list of people
- Locally improve the initial “greedy” solution by swapping group members
- Seek further improvement through local search metaheuristics
  - What are the neighbors of an assignment?
  - How can two assignments combine to create a better one?
Balanced Assignment

Model-Based Approach

Formulate a “minimal sameness” model

- Define decision variables for assignment of people to groups
  - \( x_{ij} = 1 \) if person 1 assigned to group \( j \)
  - \( x_{ij} = 0 \) otherwise

- Specify valid assignments through constraints on the variables

- Formulate sameness as an objective to be minimized
  - Total sameness = sum of the sameness of all groups

Send to an off-the-shelf solver

- Choice of excellent linear-quadratic mixed-integer solvers
- Zero-one optimization is a special case
Given
\[ P \quad \text{set of people} \]
\[ C \quad \text{set of categories of people} \]
\[ t_{ik} \quad \text{type of person } i \text{ within category } k, \text{ for all } i \in P, k \in C \]

and
\[ G \quad \text{number of groups} \]
\[ g_{\text{min}} \quad \text{lower limit on people in a group} \]
\[ g_{\text{max}} \quad \text{upper limit on people in a group} \]

Define
\[ s_{i_1i_2} = |\{k \in C: t_{i_1k} = t_{i_2k}\}|, \text{ for all } i_1 \in P, i_2 \in P \]

\text{sameness of persons } i_1 \text{ and } i_2
Model-Based Formulation (cont’d)

**Determine**

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

**To minimize**

\[ \sum_{i_1 \in P} \sum_{i_2 \in P} s_{i_1i_2} \sum_{j=1}^{G} x_{i_1j} x_{i_2j} \]

*total sameness of all pairs of people in all groups*

**Subject to**

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

*each person must be assigned to one group*

\[ g_{\text{min}} \leq \sum_{i \in P} x_{ij} \leq g_{\text{max}}, \text{ for each } j = 1, \ldots, G \]

*each group must be assigned an acceptable number of people*
Model-Based Solution

Optimize with an off-the-shelf solver

Choose among many alternatives

- Linearize and send to a mixed-integer linear solver
  * CPLEX, Gurobi, Xpress; CBC, MIPCL, SCIP
- Send quadratic formulation to a mixed-integer solver that automatically linearizes products involving binary variables
  * CPLEX, Gurobi, Xpress
- Send quadratic formulation to a nonlinear solver
  * Mixed-integer nonlinear: Knitro, BARON
  * Continuous nonlinear (might come out integer): MINOS, Ipopt, ...
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 solution approach**
- Total variation is less suitable to a greedy algorithm
- Total variation is harder to locally improve
- Client constraints are challenging to enforce

**Update or re-implement the method**
- Even small changes to the problem can necessitate major changes to the method and its implementation
Balanced Assignment

**Model-Based (cont’d)**

Replace the objective

Formulate additional constraints

Send back to the solver
Balanced Assignment

Model-Based (cont’d)

To write new objective, add variables

\[ y_{kl}^{\min} \] fewest people of category \( k \), type \( l \) in any group,

\[ y_{kl}^{\max} \] most people of category \( k \), type \( l \) in any group,

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

Add defining constraints

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

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

Minimize total variation

\[ \sum_{k \in C} \sum_{l \in T_k} (y_{kl}^{\max} - y_{kl}^{\min}) \]
Balanced Assignment

Model-Based (cont’d)

To express client requirement for women in a group, let

\[ Q = \{ i \in P : t_{i,m/f} = \text{female} \} \]

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

\[ Q = \{i \in P : t_{i,m/f} = \text{female}\} \]

Define logic variables

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

\[ = 0 \text{ otherwise, for all } j = 1, \ldots, G \]

Add constraints relating logic variables to assignment variables

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

\[ z_j = 1 \Rightarrow \sum_{i \in Q} x_{ij} \geq 2, \text{ for each } j = 1, \ldots, G \]
Model-Based (cont’d)

To express client requirement for women in a group, let

\[ Q = \{i \in P: t_{i,m/f} = \text{female}\} \]

Define logic variables

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

Linearize constraints relating logic variables to assignment variables

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

Model-Based (cont’d)

To express client requirements for group-mates, let

\[ R_{l_1l_2} = \{ i \in P : t_{i,\text{loc}} = l_1, t_{i,\text{rank}} = l_2 \}, \text{ for all } l_1 \in T_{\text{loc}}, l_2 \in T_{\text{rank}} \]

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

Add constraints

\[ \sum_{i \in R_{l_1l_2}} x_{ij} = 0 \text{ or } \sum_{i \in R_{l_1l_2}} x_{ij} + \sum_{l \in A_l} \sum_{i \in R_{l_1l}} x_{ij} \geq 2, \]

for each \( l_1 \in T_{\text{loc}}, l_2 \in T_{\text{rank}}, j = 1, \ldots, G \)
Balanced Assignment

Model-Based (cont’d)

To express client requirements for group-mates, let

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

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

Define logic variables

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

Add constraints relating logic variables to assignment variables

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

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

for each \( l_1 \in T_{\text{loc}}, l_2 \in T_{\text{rank}}, j = 1, \ldots, G \)
Balanced Assignment

Model-Based (cont’d)

To express client requirements for group-mates, let

$$R_{l_1l_2} = \{i \in P: t_{i,\text{loc}} = l_1, t_{i,\text{rank}} = l_2\}, \text{ for all } l_1 \in T_{\text{loc}}, l_2 \in T_{\text{rank}}$$

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

Define logic variables

$$w_{l_1l_2j} \in \{0,1\} = 1 \text{ if group } j \text{ has anyone from location } l_1 \text{ of rank } l_2$$

$$= 0 \text{ otherwise, for all } l_1 \in T_{\text{loc}}, l_2 \in T_{\text{rank}}, j = 1,\ldots,G$$

Linearize constraints relating logic variables to assignment variables

$$w_{l_1l_2j} \leq \sum_{i \in R_{l_1l_2}} x_{ij} \leq |R_{l_1l_2}| w_{l_1l_2j},$$

$$\sum_{i \in R_{l_1l_2}} x_{ij} + \sum_{l \in A_l} \sum_{i \in R_{l_1l}} x_{ij} \geq 2w_{l_1l_2j},$$

for each $$l_1 \in T_{\text{loc}}, l_2 \in T_{\text{rank}}, j = 1,\ldots,G$$
Method-Based Remains Popular for . . .

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

**Situations hard to formulate mathematically**
- Difficult combinatorial constraints
- Black-box objectives and constraints

**Large-scale, intensive applications**
- Routing fleets of delivery trucks
- Finding shortest routes in mapping apps
- Deep learning for facial recognition
Model-Based Has Become Common 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, . . .

**Diverse fields**
- Operations research & management science
- Business analytics
- Engineering & science
- Economics & finance
Model-Based Has Become Standard for . . .

Diverse industries

Diverse fields

Diverse kinds of users

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

These have in common . . .

- Good algebraic formulations for off-the-shelf solvers
- Users focused on modeling
Trends Favor Model-Based Optimization

*Model-based approaches have spread*
- Model-based metaheuristics (“Matheuristics”)
- Solvers for SAT, planning, *constraint programing*

*Off-the-shelf optimization solvers have kept improving*
- Solve the same problems faster and faster
- Handle broader problem classes
- Recognize special cases automatically

*Optimization models have become easier to embed within broader methods*
- Solver callbacks
- Model-based evolution of solver APIs
- APIs for optimization modeling systems
Modeling Languages for Model-Based Optimization

Background
- The modeling lifecycle
- Matrix generators
- Modeling languages

Algebraic modeling languages
- Design approaches: declarative, executable
- Example: AMPL vs. gurobipy
- Survey of available software

Balanced assignment model in AMPL
- Formulation
- Solution
The Optimization Modeling Lifecycle

1. Communicate with Client
2. Build Model
3. Prepare Data
4. Generate Optimization Problem
5. Submit Problem to Solver
6. Report & Analyze Results
Managing the Modeling Lifecycle

Goals for optimization software

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

Complication: two forms of an optimization problem

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

Challenge: translate between these two forms
Matrix Generators

Write a program to generate the solver’s form
- Read data and compute objective & constraint coefficients
- Communicate with the solver via its API
- Convert the solver’s solution for viewing or processing

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

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

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 ...

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.

Algebraic Modeling Languages

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

Advantages
  - Familiar
  - Powerful
  - Proven
Overview

Design alternatives

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

Design comparison

- Executable versus declarative using one simple example

Survey

- Solver-independent vs. solver-specific
- Proprietary vs. free
- Notable specific features
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
Algebraic Modeling Languages

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
- Can provide APIs for many popular programming languages
Comparison: Executable vs. Declarative

Two representative widely used systems

- Executable: *gurobipy*
  - Python modeling interface for Gurobi solver
  - http://gurobi.com

- Declarative: *AMPL*
  - Specialized modeling language with multi-solver support
  - http://ampl.com
Comparison

Data

gurobipy

- Assign values to Python lists and dictionaries

```
commodities = ['Pencils', 'Pens']
nodes = ['Detroit', 'Denver', 'Boston', 'New York', 'Seattle']
arcs, capacity = multidict({
    ('Detroit', 'Boston'): 100,
    ('Detroit', 'New York'): 80,
    ('Detroit', 'Seattle'): 120,
    ('Denver', 'Boston'): 120,
    ('Denver', 'New York'): 120,
    ('Denver', 'Seattle'): 120 })
```

- Provide data later in a separate file

AMPL

- Define symbolic model sets and parameters

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

set COMMODITIES := Pencils Pens;
set NODES := Detroit Denver
            Boston 'New York' Seattle;
param: ARCS: capacity:
            Boston 'New York' Seattle :=
            Detroit 100 80 120
            Denver 120 120 120;
```
### Data (cont’d)

#### gurobipy

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>60</td>
<td>-50</td>
<td>-50</td>
<td>-10</td>
<td>60</td>
<td>40</td>
<td>-40</td>
<td>-30</td>
<td>-30</td>
</tr>
</tbody>
</table>

#### AMPL

```AMPL
param inflow {COMMODITIES,NODES};

param inflow (tr):  
Pencils Pens :=
Detroit 50 60
Denver 60 40
Boston -50 -40
'New York' -50 -30
Seattle -10 -30 ;
```
Comparison

Data (cont’d)

gurobipy

cost = {
    ('Pencils', 'Detroit', 'Boston'): 10,
    ('Pencils', 'Detroit', 'New York'): 20,
    ('Pencils', 'Detroit', 'Seattle'): 60,
    ('Pencils', 'Denver', 'Boston'): 40,
    ('Pencils', 'Denver', 'New York'): 40,
    ('Pencils', 'Denver', 'Seattle'): 30,
    ('Pens', 'Detroit', 'Boston'): 20,
    ('Pens', 'Detroit', 'New York'): 20,
    ('Pens', 'Detroit', 'Seattle'): 80,
    ('Pens', 'Denver', 'Boston'): 60,
    ('Pens', 'Denver', 'New York'): 70,
    ('Pens', 'Denver', 'Seattle'): 30
}
Comparison

Data (cont’d)

AMPL

| param cost {COMMODITIES, ARCS} >= 0; |
|--|--|--|--|
| [Pencils,*,*] (tr) Detroit Denver := |
| Boston 10 40 |
| 'New York' 20 40 |
| Seattle 60 30 |
| [Pens,*,*] (tr) Detroit Denver := |
| Boston 20 60 |
| 'New York' 20 70 |
| Seattle 80 30 |
**Comparison**

**Model**

**gurobipy**

```python
m = Model('netflow')
flow = m.addVars(commodities, arcs, obj=cost, name="flow")
m.addConstrs(
    (flow.sum('*',i,j) <= capacity[i,j] for i,j in arcs), "cap")
m.addConstrs(
    (flow.sum(h,'*',j) + inflow[h,j] == flow.sum(h,j,'*')
     for h in commodities for j in nodes), "node")
```

```python
for i,j in arcs:
    m.addConstr(
        sum(flow[h,i,j] for h in commodities) <= capacity[i,j],
        "cap[%s,%s]" % (i,j))
m.addConstrs(
    (quicksum(flow[h,i,j] for i,j in arcs.select('*','j')) + inflow[h,j] ==
     quicksum(flow[h,j,k] for j,k in arcs.select(j,'*'))
     for h in commodities for j in nodes), "node")
```
Comparison

(Note on Summations)

gurobipy quicksum

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

**quicksum**( data )

A version of the Python sum function that is much more efficient for building large Gurobi expressions (LinExpr or QuadExpr objects). The function takes a list of terms as its argument.

Note that while quicksum is much faster than sum, it isn’t the fastest approach for building a large expression. Use addTerms or the LinExpr() constructor if you want the quickest possible expression construction.
Comparison

Model (cont’d)

AMPL

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

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

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

subject to Conservation {h in COMMODITIES, j in NODES}:
    sum {(i,j) in ARCS} Flow[h,i,j] + inflow[h,j] =
    sum {(j,i) in ARCS} Flow[h,j,i];
```
**Comparison**

**Solution**

```python
m.optimize()

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

Solution (cont’d)

AMPL

```ampl
AMPL: solve;
Gurobi 8.0.0: optimal solution; objective 5500
2 simplex iterations

AMPL: display Flow;

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

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

;
Comparison

Integration with Solvers

gurobipy

- Works closely with the Gurobi solver: callbacks during optimization, fast re-solves after problem changes
- Offers convenient extended expressions: min/max, and/or, if-then-else

AMPL

- Supports all popular solvers
- Extends to general nonlinear and logic expressions
  - Connects to nonlinear function libraries and user-defined functions
- Automatically computes nonlinear function derivatives
Comparison

Integration with Applications

gurobipy

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

AMPL

- Modeling language extended with loops, tests, assignments
- Application programming interfaces (APIs) for calling AMPL from C++, C#, Java, MATLAB, Python, R
  - Efficient methods for data interchange
- Add-ons for streamlined deployment
  - QuanDec by Cassotis
  - Opalytics Cloud Platform
Algebraic Modeling Languages

Software Survey

Solver-specific
- Associated with popular commercial solvers
- Executable and declarative alternatives

Solver-independent
- Support multiple solvers and solver types
- Mostly commercial/declarative and free/executable
Survey

Solver-Specific

Declarative, commercial

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

Executable, commercial

- Concert Technology C++ for CPLEX
- gurobipy for Gurobi
- sasoptpy for SAS Optimization
Survey

Solver-Independent

Declarative, commercial
- AIMMS
- AMPL
- GAMS
- MPL

Declarative, free
- CMPL
- GMPL / MathProg

Executable, free
- PuLP; Pyomo / Python
- YALMIP; CVX / MATLAB
- JuMP / Julia
- FLOPC++ / C++
Algebraic Modeling Languages

Trends

Commercial, declarative modeling systems
- Established lineup of solver-independent modeling systems that represent decades of development and support
- Extended with scripting, APIs, data tools to promote integration with broader applications

Commercial, executable modeling systems
- Increasingly essential to commercial solver offerings
- Becoming the recommended APIs for solvers

Free, executable modeling systems
- A major current focus of free optimization software development
- Interesting new executable modeling languages have become easier to develop than interesting new solvers
Notable cases not detailed earlier . . .

- **AIMMS (solver-independent, commercial, declarative)** has extensive application development tools built in
- **CMPL (solver-independent, free, declarative)** has an IDE, Python and Java APIs, and remote server support
- **GMPL/MathProg (solver-independent, free, declarative)** is a free implementation of mainly a subset of AMPL
- **JuMP (solver-independent, free, executable)** claims greater efficiency through use of a new programming language, Julia
- **MOSEL for Xpress (solver-specific, commercial)** a hybrid of declarative and executable, has recently been made free and may accept other solvers
Balanced Assignment Revisited

Given

- $P$ set of people
- $C$ set of categories of people
- $t_{ik}$ type of person $i$ within category $k$, for all $i \in P, k \in C$

and

- $G$ number of groups
- $g_{\text{min}}$ lower limit on people in a group
- $g_{\text{max}}$ upper limit on people in a group

Define

- $T_k = \bigcup_{i \in P} \{t_{ik}\}$, for all $k \in C$
  - set of all types of people in category $k$
Balanced Assignment Revisited in AMPL

Sets, parameters

set PEOPLE;   # individuals to be assigned
set CATEG;
param type {PEOPLE,CATEG} symbolic;
    # categories by which people are classified;
    # type of each person in each category
param numberGrps integer > 0;
param minInGrp integer > 0;
param maxInGrp integer >= minInGrp;
    # number of groups; bounds on size of groups
set TYPES {k in CATEG} = setof {i in PEOPLE} type[i,k];
    # all types found in each category
Balanced Assignment

Determine

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

\[ y_{kl}^{\min} \] fewest people of category \( k \), type \( l \) in any group,
\[ y_{kl}^{\max} \] most people of category \( k \), type \( l \) in any group,

for each \( k \in C, l \in T_k \)

Where

\[ y_{kl}^{\min} \leq \sum_{i \in P : t_{ik} = l} x_{ij}, \text{ for each } j = 1, \ldots, G; \ k \in C, l \in T_k \]
\[ y_{kl}^{\max} \geq \sum_{i \in P : t_{ik} = l} x_{ij}, \text{ for each } j = 1, \ldots, G; \ k \in C, l \in T_k \]
Balanced Assignment *in AMPL*

**Variables, defining constraints**

```AMPL
var Assign {i in PEOPLE, j in 1..numberGrps} binary;
    # Assign[i,j] is 1 if and only if
    # person i is assigned to group j

var MinType {k in CATEG, TYPES[k]};
var MaxType {k in CATEG, TYPES[k]};
    # fewest and most people of each type, over all groups

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

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

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

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

Minimize

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

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

Subject to

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

each person must be assigned to one group

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

each group must be assigned an acceptable number of people
Balanced Assignment in AMPL

Objective, assignment constraints

minimize TotalVariation:
    sum {k in CATEG, l in TYPES[k]} (MaxType[k,l] - MinType[k,l]);
    # Total variation over all types

subj to AssignAll {i in PEOPLE}:
    sum {j in 1..numberGrps} Assign[i,j] = 1;
    # Each person must be assigned to one group

subj to GroupSize {j in 1..numberGrps}:
    minInGrp <= sum {i in PEOPLE} Assign[i,j] <= maxInGrp;
    # Each group must have an acceptable size

\[ g_{\min} \leq \sum_{i \in P} x_{ij} \leq g_{\max}, \text{ for each } j = 1, \ldots, G \]
Balanced Assignment

Define also

\[ Q = \{ i \in P : t_{i,m/f} = \text{female} \} \]

Determine

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

\[ = 0 \text{ otherwise, for all } j = 1, \ldots, G \]

Subject to

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

\[ \text{each group must have either} \]

\[ \text{no women } (z_j = 0) \text{ or } \geq 2 \text{ women } (z_j = 1) \]
Balanced Assignment in AMPL

Supplemental constraints

\[
\text{set WOMEN = \{i in PEOPLE: type[i,'m/f'] = 'F'\};}
\]
\[
\text{var WomenInGroup \{j in 1..numberGrps\} binary;}
\]
\[
\text{subj to Min2WomenInGroupLO \{j in 1..numberGrps\}:}
\]
\[
2 \times \text{WomenInGroup}[j] \leq \text{sum \\{i in WOMEN\} Assign[i,j];}
\]
\[
\text{subj to Min2WomenInGroupUP \{j in 1..numberGrps\}:}
\]
\[
\text{sum \\{i in WOMEN\} Assign[i,j] \leq \text{card(WOMEN)} \times \text{WomenInGroup}[j];}
\]

\[2z_j \leq \sum_{i\in Q} x_{ij} \leq |Q| z_j, \text{ for each } j = 1, \ldots, G\]
**Balanced Assignment**

**Modeling Language Data**

210 people

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

Modeling Language Data

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

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

param type:
    dept   loc     'm/f'   title   :=
    BIW   NNE   Peoria        M   Assistant
    KRS   WSW   Springfield   F   Assistant
    TLR   NNW   Peoria        F   Adjunct
    VAA   NNW   Peoria        M   Deputy
    JRT   NNE   Springfield   M   Deputy
    AMR   SSE   Peoria        M   Deputy
    MES   NNE   Peoria        M   Consultant
    JAD   NNE   Peoria        M   Adjunct
    MJR   NNE   Springfield   M   Assistant
    JRS   NNE   Springfield   M   Assistant
    HCN   SSE   Peoria        M   Deputy
    DAN   NNE   Springfield   M   Adjunct

......

param numberGrps := 12 ;
param minInGrp := 16 ;
param maxInGrp := 19 ;
```
Model-Based Optimization

Balanced Assignment

Modeling Language Solution

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

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

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

2568 variables:
   2532 binary variables
   36 linear variables

678 constraints, all linear; 26328 nonzeros
   210 equality constraints
   456 inequality constraints
   12 range constraints

1 linear objective; 36 nonzeros.

CPLEX 12.8.0.0: optimal integer solution; objective 16
115096 MIP simplex iterations
1305 branch-and-bound nodes

10.5 sec
Balanced Assignment

Modeling Language Solution

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

```ampl
ampl: model BalAssign.mod;
ampl: data BalAssign.dat;
ampl: option solver gurobi;
ampl: option show_stats 1;
ampl: solve;
```

2568 variables:
- 2532 binary variables
- 36 linear variables

678 constraints, all linear; 26328 nonzeros
- 210 equality constraints
- 456 inequality constraints
- 12 range constraints

1 linear objective; 36 nonzeros.

Gurobi 8.0.0: optimal solution; objective 16
483547 simplex iterations
808 branch-and-cut nodes

108.8 sec
Balanced Assignment (logical)

Define also

\[ Q = \{ i \in P : t_{i,m/f} = \text{female} \} \]

Determine

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

Where

\[ z_j = 0 \Rightarrow \sum_{i \in Q} x_{ij} = 0, \]
\[ z_j = 1 \Rightarrow \sum_{i \in Q} x_{ij} \geq 2, \text{ for each } j = 1, \ldots, G \]
Balanced Assignment in AMPL

Supplemental logical constraints

```
set WOMEN = {i in PEOPLE: type[i,'m/f'] = 'F'};
var WomenInGroup {j in 1..numberGrps} binary;
subj to Min2WomenInGroup {j in 1..numberGrps}:
    WomenInGroup[j] = 0 ==> sum {i in WOMEN} Assign[i,j] = 0
else sum {i in WOMEN} Assign[i,j] >= 2;
```

\[ z_j = 0 \Rightarrow \sum_{i \in Q} x_{ij} = 0, \]
\[ z_j = 1 \Rightarrow \sum_{i \in Q} x_{ij} \geq 2, \text{ for each } j = 1, \ldots, G \]
Balanced Assignment

Balanced Assignment in AMPL

Send to "linear" solver

ampl: model BalAssignWomen.mod
ampl: data BalAssign.dat
ampl: option solver gurobi;
ampl: solve

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

Gurobi 8.0.0: optimal solution; objective 16
265230 simplex iterations
756 branch-and-cut nodes

42.8 sec
Balanced Assignment

Balanced Assignment in AMPL (refined)

Add bounds on variables

```ampl
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
ampl: solve
Presolve eliminates 72 constraints.
...
Gurobi 8.0.0: optimal solution; objective 16
1617 simplex iterations
1 branch-and-cut nodes
0.16 sec
```
Nonlinear Optimization in AMPL

Example: Shekel function

Mathematical Formulation

Given

- \( m \) number of locally optimal points
- \( n \) number of variables

and

- \( a_{ij} \) for each \( i = 1, \ldots, m \) and \( j = 1, \ldots, n \)
- \( c_i \) for each \( i = 1, \ldots, m \)

Determine

- \( x_j \) for each \( j = 1, \ldots, n \)

to maximize

\[
\sum_{i=1}^{m} \frac{1}{c_i + \sum_{j=1}^{n} (x_j - a_{ij})^2}
\]
Modeling Language Formulation

*Symbolic model (AMPL)*

```AMPL
param m integer > 0;
param n integer > 0;
param a {1..m, 1..n};
param c {1..m};
var x {1..n};

maximize objective:
    sum {i in 1..m} 1 / (c[i] + sum {j in 1..n} (x[j] - a[i,j])^2);
```

\[ \sum_{i=1}^{m} \frac{1}{c_i + \sum_{j=1}^{n} (x_j - a_{ij})^2} \]
Modeling Language Data

Explicit data (independent of model)

\begin{verbatim}
param m := 5 ;
param n := 4 ;

param a: 1 2 3 4 :=
  1  4  4  4  4
  2  1  1  1  1
  3  8  8  8  8
  4  6  6  6  6
  5  3  7  3  7 ;

param c :=
  1  0.1
  2  0.2
  3  0.2
  4  0.4
  5  0.4 ;
\end{verbatim}
Modeling Language Solution

*Model + data = problem instance to be solved*

```ampl
ampl: model shekelEX.mod;
ampl: data shekelEX.dat;
ampl: option solver knitro;
ampl: solve;

Knitro 10.3.0: Locally optimal solution.
objective 5.055197729; feasibility error 0
6 iterations; 9 function evaluations

ampl: display x;
x [*] :=
 1  1.00013
 2  1.00016
 3  1.00013
 4  1.00016
;
```
Modeling Language Solution

... again with multistart option

```ampl
ampl: model shekelEX.mod;
ampl: data shekelEX.dat;
ampl: option solver knitro;
ampl: option knitro_options 'ms_enable=1 ms_maxsolves=100';
ampl: solve;
Knitro 10.3.0: Locally optimal solution.
objective 10.15319968; feasibility error 0
43 iterations; 268 function evaluations
ampl: display x;
x [*] :=
 1 4.00004
 2 4.00013
 3 4.00004
 4 4.00013
;```


Solution (cont’d)

... again with a “global” solver

```
ampl: model shekelEX.mod;
ampl: data shekelEX.dat;
ampl: option solver baron;
ampl: solve;
BARON 17.10.13 (2017.10.13):
43 iterations, optimal within tolerances.
Objective 10.15319968
ampl: display x;
x [*] :=
 1  4.00004
 2  4.00013
 3  4.00004
 4  4.00013
;
```
Solvers for Model-Based Optimization

Off-the-shelf solvers for broad problem classes

Three widely used types

- “Linear”
- “Nonlinear”
- “Global”
“Linear” Solvers

Require objective and constraint coefficients

Linear objective and constraints

- Continuous variables
  * Primal simplex, dual simplex, interior-point
- Integer (including zero-one) variables
  * Branch-and-bound + feasibility heuristics + cut generation
  * Automatic transformations to integer:
    piecewise-linear, discrete variable domains, indicator constraints
  * “Callbacks” to permit problem-specific algorithmic extensions

Quadratic extensions

- Convex elliptic objectives and constraints
- Convex conic constraints
- Variable × binary in objective
  * Transformed to linear (or to convex if binary × binary)
“Linear” Solvers (cont'd)

**CPLEX, Gurobi, Xpress**
- Dominant commercial solvers
- Similar features
- Supported by many modeling systems

**SAS Optimization, MATLAB intlinprog**
- Components of widely used commercial analytics packages
- SAS performance within 2x of the “big three”

**MOSEK**
- Commercial solver strongest for conic problems

**CBC, MIPCL, SCIP**
- Fastest noncommercial solvers
- Effective alternatives for easy to moderately difficult problems
- MIPCL within 7x on some benchmarks
“Nonlinear” Solvers

Require function and derivative evaluations

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

Extension to integer variables
“Nonlinear” Solvers

Knitro

- Most extensive commercial nonlinear solver
- Choice of methods; automatic choice of multiple starting points
- Parallel runs and parallel computations within methods
- Continuous and integer variables

CONOPT, LOQO, MINOS, SNOPT

- Highly regarded commercial solvers for continuous variables
- Implement a variety of methods

Bonmin, Ipopt

- Highly regarded free solvers
  - Ipopt for continuous problems via interior-point methods
  - Bonmin extends to integer variables
“Global” Solvers

Require expression graphs (or equivalent)

Nonlinear + global optimality

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

BARON

- Dominant commercial global solver

Couenne

- Highly regarded noncommercial global solver

LGO

- High-quality solutions, may be global
- Objective and constraint functions may be nonsmooth
Off-the-Shelf Solvers

Benchmarks

Prof. Hans Mittelmann’s benchmark website

DECISION TREE FOR OPTIMIZATION SOFTWARE

BENCHMARKS FOR OPTIMIZATION SOFTWARE

By Hans Mittelmann (mittelmann at asu.edu)

Note that on top of the benchmarks a link to logfiles is given!

NOTE ALSO THAT WE DO NOT USE PERFORMANCE PROFILES. SEE THIS PAPER

WE USE INSTEAD THE SHIFTED GEOMETRIC MEAN
**Off-the-Shelf Solvers**

**Benchmarks**

*By problem type and test set*

- **MIXED INTEGER LINEAR PROGRAMMING**
  - MILP Benchmark - MIPLIB2010 (4-25-2018)
  - The Solvable MIPLIB Instances (4-28-2018) (MIPLIB2010)
  - MILP cases that are slightly pathological (4-25-2018)
  - Feasibility Benchmark (4-25-2018) (MIPLIB2010)
  - Infeasibility Detection for MILP (4-25-2018) (MIPLIB2010)

- **SEMIDEFINITE/SQL PROGRAMMING**
  - SQL problems from the 7th DIMACS Challenge (8-8-2002)
  - Several SDP codes on sparse and other SDP problems (1-17 2018)
  - Infeasible SDP Benchmark (3-9-2018)
  - Large SOCP Benchmark (4-25-2018)
  - MISOCP Benchmark (4-25-2018)

- **NONLINEAR PROGRAMMING**
  - AMPL-NLP Benchmark (4-16-2018)
Off-the-Shelf Solvers

Benchmarks

Documentation, summaries, links to detailed results

The following codes were run with a limit of 2 hours on the MIPLIB2010 benchmark set with the MIPLIB2010 scripts (exc Matlab) on two platforms. 1/4 threads: Intel i7-4790K, 4 cores, 32GB, 4GHz, available memory 24GB; 12 threads: Intel Xeon X5680, 2x6 cores, 32GB, 3.33GHz, available memory 24GB. These are updated and extended versions of the results produced for the MIPLIB2010 paper.

CPLX-12.8.0: CPLEX
GUROBI-8.0.0: GUROBI
ug(Scip-cplex): 5.0.0: Parallel development version of SCIP (SCIP+CPLEX/SOPLEX on 1 thread)
CBC-2.9.8: CBC
XPRESS-8.4.0: XPRESS
MATLAB-2018a: MATLAB (intlinprog)
MIPCL-1.5.1: MIPCL
SAS-OR-14.3: SAS

Table for single thread. Result files per solver, Log files per solver

Table for 4 threads. Result files per solver, Log files per solver

Table for 12 threads. Result files per solver, Log files per solver

Statistics of the problems can be obtained from the MIPLIB2010 webpage

Unscaled and scaled shifted geometric means of run times

All non-successes are counted as max-time. The third line lists the number of problems (87 total) solved.
Curious? Try Them Out on NEOS!

NEOS Server: State-of-the-Art Solvers for Numerical Optimization

The NEOS Server is a free Internet-based service for solving numerical optimization problems. Hosted by the Wisconsin Institute for Discovery at the University of Wisconsin in Madison, the NEOS Server provides access to more than 60 state-of-the-art solvers in more than a dozen optimization categories. Solvers hosted by the University of Wisconsin in Madison run on distributed high-performance machines enabled by the HTCondor software, remote solvers run on machines at Arizona State University, the University of Klagenfurt in Austria, and the University of Minho in Portugal.

The NEOS Guide website complements the NEOS Server, showcasing optimization case studies, presenting optimization information and resources, and providing background information on the NEOS Server.
**NEOS Server**

**Solver & Language Listing**

![Solver & Language Listing](https://neos-server.org/neos/solvers/index.html)

### Linear Programming
- Cbc [AMPL Input][GAMS Input][MPS Input]
- CPLEX [AMPL Input][GAMS Input][LP Input][MPS Input][NL Input]
- cplexmp [AMPL Input][CPLEX Input][MPS Input]
- FICO-Xpress [AMPL Input][GAMS Input][MOSEL Input][MPS Input][NL Input]
- Gurobi [AMPL Input][GAMS Input][LP Input][MPS Input][NL Input]
- MINTO [AMPL Input]
- MOSK [AMPL Input][GAMS Input][LP Input][MPS Input][NL Input]
- proxy [CPLEX Input][MPS Input]
- osqp [AMPL Input][LP Input][MPS Input]
- scop [AMPL Input][CPLEX Input][GAMS Input][MPS Input][OSIL Input][ZIMPL Input]
- SYMPHONY [MPS Input]

### Mixed Integer Linear Programming
- ANTIGONE [GAMS Input]
- CONOPT [AMPL Input][GAMS Input]
- filter [AMPL Input]
- ipopt [AMPL Input][GAMS Input][NL Input]
- Knitro [AMPL Input][GAMS Input]
- LANCELOT [AMPL Input]

### Mixed Integer Nonlinearly Constrained Optimization
- CONOPT [AMPL Input][GAMS Input]
- filter [AMPL Input]
- ipopt [AMPL Input][GAMS Input][NL Input]
- Knitro [AMPL Input][GAMS Input]
- LANCELOT [AMPL Input]

### Mixed-Integer Optimal Control Problems
- ANTIGONE [GAMS Input]
- CONOPT [AMPL Input][GAMS Input]
- filter [AMPL Input]
- ipopt [AMPL Input][GAMS Input][NL Input]
- Knitro [AMPL Input][GAMS Input]
- LANCELOT [AMPL Input]
About the NEOS Server

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

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

Interfaces
- Web browser
- Special solver (“Kestrel”) for AMPL and GAMS
- Python API
About the NEOS Server (cont’d)


Limits

- 8 hours
- 3 GBytes

Operation

- Requests queued centrally, distributed to various servers for solving
- 650,000+ requests served in the past year, about 1800 per day or 75 per hour
- 17,296 requests on peak day (15 March 2018)
Constraint Programming

Between method-based and model-based

- Relies on solvers
- Focus of the work may be on methods or may be on modeling

Method-based view

- CP solver as a framework for implementation

Model-based view

- CP solver as an alternative type
Method-Based vs. Model-Based

Method-based view
- Define global constraints to express the problem
- Implement methods required to use the constraints in the solver
  - Filtering, checking, explanation, counting, reification, . . .
- Program the search procedure for the solver
  - Possibly add constraints via restrictions to the search

Model-based view
- Write a model naturally without linearizing
  - Not-linear operators: min, max, abs
  - Logic operators: and, or, not, if-then
  - Global constraints: alldiff, atleast, atmost
  - Variables as indices
- Send to an off-the-shelf CP solver
CP Approach to Balanced Assignment

Fewer variables with larger domains

```plaintext
var Assign {i in PEOPLE, j in 1..numberGrps} binary;
# Assign[i,j] is 1 if and only if
# person i is assigned to group j

var Assign {i in PEOPLE} integer >= 1, <= numberGrps;
# Assign[i] is the group to which i is assigned
```
Balanced Assignment

CP Approach

Global constraint for assignment to groups

subj to AssignAll \{i in PEOPLE\}:
  \[
  \sum_{j \in 1..\text{numberGrps}} \text{Assign}[i,j] = 1;
  \]
  # Each person assigned to one group

subj to GroupSize \{j \in 1..\text{numberGrps}\}:
  \[
  \text{minInGrp} \leq \sum_{i \in \text{PEOPLE}} \text{Assign}[i,j] \leq \text{maxInGrp};
  \]
  # Each group has an acceptable size

subj to GroupSize \{j \in 1..\text{numberGrps}\}:
  \[
  \text{minInGrp} \leq \text{numberOf} j \in (\{i \in \text{PEOPLE} \} \text{Assign}[i]) \leq \text{maxInGrp};
  \]
  # Each group has an acceptable size
**Balanced Assignment**

**CP Approach**

**Disjunctive constraint for women in a group**

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

subj to Min2WomenInGroupUP {j in 1..numberGrps}:
    sum {i in WOMEN} Assign[i,j] <= card(WOMEN) * WomenInGroup[j];
    # Number of women in each group is either
    # 0 (WomenInGroup[j] = 0) or >= 2 (WomenInGroup[j] = 1)

subj to Min2WomenInGroupL {j in 1..numberGrps}:
    numberof j in ({i in WOMEN} Assign[i]) = 0 or
    numberof j in ({i in WOMEN} Assign[i]) >= 2;
    # Number of women in each group is either 0 or >= 2
```
**Balanced Assignment**

**CP Approach**

**Solve with IBM ILOG CP**

```plaintext
ampl: model BalAssign+CP.mod
ampl: data BalAssign.dat
ampl: option solver ilogcp;
ampl: solve;

246 variables:
   36 integer variables
   210 nonlinear variables
444 algebraic constraints, all nonlinear; 23112 nonzeros
   432 inequality constraints
   12 range constraints
12 logical constraints
1 linear objective; 28 nonzeros.

ilogcp 12.7.0: optimal solution
512386 choice points, 232919 fails, objective 16

5.3 sec
```
Algebraic Modeling Languages for CP

**IBM CPLEX C++ API**
- Executable, solver-specific

**IBM CPLEX OPL**
- Declarative, solver-specific

**MiniZinc**
- Declarative, solver-independent
Summary: Model-Based Optimization

Division of labor

- Analysts who build symbolic optimization models
- Developers who create general-purpose solvers

A successful approach across very diverse application areas

Modeling languages (like AMPL) bridge the gap between models and solvers

- Translate between modeler’s form and algorithm’s form
- Maintain independence of model and data
- Offer independence of model and data from solver