Model-Based Optimization
+ Application Programming
= Streamlined Deployment in AMPL

Robert Fourer, Filipe Brandão
{4er, fdabrandao}@ampl.com
AMPL Optimization Inc.
www.ampl.com — +1 773-336-AMPL

INFORMS Annual Meeting
Phoenix— 4-7 November 2018
Session MD34, Technology Tutorials
Examples

Model-based optimization
- Model-based vs. Method-based approaches
  * Example: Balanced assignment
- Declarative vs. Executable modeling
  * Example: AMPL vs. gurobipy for multicommodity flow

Application programming
- Extending a modeling language with scripting
  * Example: Tradeoffs between cutting-stock objectives

Streamlined deployment
- Modeling language APIs
  * Example: Pattern generation in Python and R
- Modeling language extensions
  * Examples: Embedded Python for AMPL (a preview)
Model-Based vs. Method-Based Approaches to Optimization

Example: Balanced Assignment

- 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
**Balanced Assignment**

**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 \( p \)
    \( s_{\text{Min}} = \infty \)
    Repeat for each group \( G \)
        \( s = \text{total "sameness" in } G \cup \{p\} \)
        if \( s < s_{\text{Min}} \) then
            \( s_{\text{Min}} = s \)
            \( G_{\text{Min}} = G \)
    Assign person \( p \) to group \( G_{\text{Min}} \)
```
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
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
- \( g_{\text{min}} \) lower limit on people in a group
- \( g_{\text{max}} \) upper limit on people in a group

Define

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

sameness of persons \( i_1 \) and \( i_2 \)
Balanced Assignment

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

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

total sameness of all pairs of people in all groups

Subject to

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

each person must be assigned to one group

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

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

Model-Based Solution

Optimize with 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, . . .
Balanced Assignment

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
Balanced Assignment

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

\[
\begin{align*}
\gamma_{kl}^{\min} & \text{ fewest people of category } k, \text{ type } l \text{ in any group}, \\
\gamma_{kl}^{\max} & \text{ most people of category } k, \text{ type } l \text{ in any group},
\end{align*}
\]

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

Add defining constraints

\[
\begin{align*}
\gamma_{kl}^{\min} & \leq \sum_{i \in P: t_{ik} = l} x_{ij}, \text{ for each } j = 1, \ldots, G; \ k \in C, l \in T_k \\
\gamma_{kl}^{\max} & \geq \sum_{i \in P: t_{ik} = l} x_{ij}, \text{ for each } j = 1, \ldots, G; \ k \in C, l \in T_k
\end{align*}
\]

Minimize total variation

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

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

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

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
- Modeling languages
- Algebraic modeling languages

**Design approaches**

- Declarative vs. executable modeling languages
- Example: AMPL vs. gurobipy

**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**
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

- Separate language to be learned
- Overhead in translation to algorithm’s form
- Confidential formulation to be protected

Great advantages

- Faster modeling cycles
- More reliable modeling
- More maintainable applications
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
Algebraic modeling language and system

- Built specially for optimization
- Designed to support many solvers

Options for deployment

- Scripting based on modeling language extensions
- APIs for C++, C#, Java, MATLAB, Python, R
- Embedded Python processed by the Python API
  * (available soon)

Application-building toolkits (not covered in this talk)

- QuanDec / built on Java API
- Opalytics (Accenture) / connected via Python API
Executable vs. Declarative Modeling Languages for Optimization

**Example: Multicommodity Flow**
- ship multiple goods over a network

**Given**
- networks nodes and arc
- supplies or demands at the nodes
- capacities on the arcs

**Determine**
- how much to ship over each arc

**So that**
- demands are met by the supplies
- *shipping costs are minimized*
**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
Algebraic Modeling 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

```python
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

```plaintext
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 ;
```
Comparison

Data (cont’d)

gurobipy

\[
\text{inflow} = \{
\begin{align*}
('Pencils', 'Detroit') & : 50, \\
('Pencils', 'Denver') & : 60, \\
('Pencils', 'Boston') & : -50, \\
('Pencils', 'New York') & : -50, \\
('Pencils', 'Seattle') & : -10, \\
('Pens', 'Detroit') & : 60, \\
('Pens', 'Denver') & : 40, \\
('Pens', 'Boston') & : -40, \\
('Pens', 'New York') & : -30, \\
('Pens', 'Seattle') & : -30
\end{align*}
\}
\]

AMPL

\[
\text{param inflow \{COMMODITIES, NODES\};}
\]

\[
\text{param inflow (tr):}
\begin{align*}
Pencils & \\
Pens & \\
\begin{array}{ll}
\text{Detroit} & 50 \\
\text{Denver} & 60 \\
\text{Boston} & -50 \\
'New York' & -50 \\
\text{Seattle} & -10
\end{array}
\end{align*}
\]

\[
\text{Pens :=}
\begin{align*}
Pencils & \\
Pens & \\
\begin{array}{ll}
\text{Detroit} & 60 \\
\text{Denver} & 40 \\
\text{Boston} & -40 \\
'New York' & -30 \\
\text{Seattle} & -30
\end{array}
\end{align*}
\]
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;

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

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")

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

gurobipy

```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
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**

\[ T_k = \bigcup_{i \in P} \{ t_{ik} \}, \text{ for all } k \in C \]

set of all types of people in category \( k \)
Balanced Assignment Revisited \textit{in AMPL}

\textbf{Sets, parameters}

\begin{verbatim}
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
\end{verbatim}
**Balanced Assignment**

**Determine**

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

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

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

**Where**

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

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

each group must have either

no women \((z_j = 0)\) or \(\geq 2\) women \((z_j = 1)\)
Balanced Assignment in AMPL

Supplemental constraints

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

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

Modeling Language Data

210 people

<table>
<thead>
<tr>
<th>set PEOPLE :=</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIW  AJH  FWI  IGN  KWR  KKI  HMN  SML  RSR  TBR</td>
</tr>
<tr>
<td>KRS  CAE  MPO  CAR  PSL  BCG  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  MMY  API  ASA  JLN</td>
</tr>
<tr>
<td>JRT  SJO  WMS  RLN  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  RJY  TWD  MIA  JFR  LHS</td>
</tr>
<tr>
<td>JAD  CWU  PMY  CAH  SJH  EGR  JMQ  GGH  MMH  JWR</td>
</tr>
<tr>
<td>MJR  EAZ  WAD  LVN  DHR  ABE  LSR  MBT  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  KWA  MLE  FJL  ESF  TJJ  WHF</td>
</tr>
<tr>
<td>TANG  FEE  MTH  RMN  WFS  CEH  SOL  ASO  MDI  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  RJE  MRZ  MRK  XYF</td>
</tr>
<tr>
<td>JCO  PSN  SCS  RDL  TMN  CGY  GMR  SER  RMS  JEN</td>
</tr>
<tr>
<td>DWO  REN  DGR  DET  FJT  RJZ  MBY  RSN  REZ  BLW</td>
</tr>
</tbody>
</table>
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 ;
Balanced Assignment

Modeling Language Solution

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

```ampl
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
Extending a Modeling Language with Scripting

Example: Roll Cutting

- fill orders for rolls of various widths

Given

- raw rolls of a large (fixed) width
- demands for various (smaller) ordered widths
- a selection of cutting patterns that may be used

Determine

- the number of times to cut each pattern

So that

- demands are met (or slightly exceeded)
- raw rolls cut and wasted material are minimized
**AMPL Model**

**Mathematical Formulation**

*Given*

\[ w \] width of “raw” rolls

\[ W \] set of (smaller) ordered widths

\[ n \] number of cutting patterns considered

*and*

\[ a_{ij} \] occurrences of width \( i \) in pattern \( j \),

for each \( i \in W \) and \( j = 1, \ldots, n \)

\[ b_i \] orders for width \( i \), for each \( i \in W \)

\[ o \] limit on overruns
**AMPL Model**

**Mathematical Formulation (cont’d)**

**Determine**

\[ X_j \] number of rolls to cut using pattern \( j \),
for each \( j = 1, \ldots, n \)

**to minimize**

\[ \sum_{j=1}^{n} X_j \]

total number of rolls cut

**subject to**

\[ b_i \leq \sum_{j=1}^{n} a_{ij} X_j \leq b_i + o, \text{ for all } i \in W \]

number of rolls of width \( i \) cut
must be at least the number ordered,
and must be within the overrun limit
AMPL Formulation

Symbolic model

```
param rawWidth;
set WIDTHS;

param nPatterns integer > 0;
set PATTERNS = 1..nPatterns;

param rolls {WIDTHS,PATTERNS} >= 0, default 0;
param order {WIDTHS} >= 0;
param overrun;

var Cut {PATTERNS} integer >= 0;

minimize TotalCut: sum {p in PATTERNS} Cut[p];

subject to OrderLimits {w in WIDTHS}:
    order[w] <= sum {p in PATTERNS} rolls[w,p] * Cut[p] <= order[w] + overrun;
```

\[ b_i \leq \sum_{j=1}^{n} a_{ij} X_j \leq b_i + o \]
**AMPL Model**

**AMPL Formulation (cont’d)**

**Explicit data (independent of model)**

```AMPL
param rawWidth := 64.5 ;
param: WIDTHS: order :=
   6.77    10
   7.56    40
   17.46    33
   18.76    10 ;
param nPatterns := 9 ;
param rolls:  1  2  3  4  5  6  7  8  9 :=
   6.77   0  1  1  0  3  2  0  1  4
   7.56   1  0  2  1  1  4  6  5  2
  17.46   0  1  0  2  1  0  1  1  1
  18.76   3  2  2  1  1  0  0  0  0 ;
param overrun := 6 ;
```
*AMPL Model*

**AMPL Command Language**

*Model + data = problem instance to be solved*

```
ampl: model cut.mod;
ampl: data cut.dat;
ampl: option solver cplex;
ampl: solve;
CPLEX 12.8.0.0: optimal integer solution; objective 20
3 MIP simplex iterations
0 branch-and-bound nodes
ampl: option omit_zero_rows 1;
ampl: option display_1col 0;
ampl: display Cut;
4 13 8 5 9 2
```
Command Language (cont’d)

Solver choice independent of model and data

```ampl
ampl: model cut.mod;
ampl: data cut.dat;
ampl: option solver gurobi;
ampl: solve;
Gurobi 8.0.0: optimal solution; objective 20
7 simplex iterations
1 branch-and-cut nodes
ampl: option omit_zero_rows 1;
ampl: option display_1col 0;
ampl: display Cut;
2 1 4 13 8 5 9 1
```
AMPL Model

Command Language (cont’d)

Results available for browsing

ampl: display {p in PATTERNS} sum {w in WIDTHS} w * rolls[w,p];
1 63.84   3 59.41   5 64.09   7 62.82   9 59.66   # material used
2 61.75   4 61.24   6 62.54   8 62.0              # in each pattern

ampl: display sum {p in PATTERNS}
ampl:  Cut[p] * (rawWidth - sum {w in WIDTHS} w * rolls[w,p]);
62.32                                             # total waste
# in solution

ampl: display OrderLimits.lslack;
   6.77  0                                          # overruns
   7.56  0                                          # of each pattern
   17.46 0
   18.76 5
**AMPL Script**

*Trade off two objectives*

- Minimize rolls cut
  - Fewer overruns, possibly more waste
- Minimize waste
  - Less waste, possibly more overruns

```AMPL
minimize TotalCut:
    sum {p in PATTERNS} Cut[p];

minimize TotalWaste:
    sum {p in PATTERNS} 
        Cut[p] * (rawWidth - sum {w in WIDTHS} w * rolls[w,p]);
```
**AMPL Script**

**Parametric Analysis of Tradeoff**

**Minimize rolls cut**
- Set large overrun limit in data

**Minimize waste**
- Reduce overrun limit 1 roll at a time
- If there is a change in number of rolls cut
  - record total waste (increasing)
  - record total rolls cut (decreasing)
- Stop when no further progress possible
  - problem becomes infeasible *or*
  - total rolls cut falls to the minimum
- Report table of results
Parametric Analysis (cont’d)

Script (setup and initial solve)

```AMPL
model cutTradeoff.mod;
data cutTradeoff.dat;
set OVER default {} ordered by reversed Integers;
param minCut;
param minCutWaste;
param minWaste {OVER};
param minWasteCut {OVER};
param prev_cut default Infinity;
option solver gurobi;
option solver_msg 0;
objective TotalCut;
solve >Nul;
let minCut := TotalCut;
let minCutWaste := TotalWaste;
objective TotalWaste;
```
Parametric Analysis (cont’d)

Script (looping and reporting)

```AMPL
for {k in overrun .. 0 by -1} {
    let overrun := k;
    solve >Nul;
    if solve_result = 'infeasible' then break;
    if TotalCut < prev_cut then {
        let OVER := OVER union {k};
        let minWaste[k] := TotalWaste;
        let minWasteCut[k] := TotalCut;
    }
    if TotalCut = minCut then break;
}
printf 'Min%3d rolls with waste%6.2f\n\n', minCut, minCutWaste;
printf ' Over Waste Cut\n';
printf {k in OVER}: '%4d%8.2f%5d\n', k, minWaste[k], minWasteCut[k];
```
**AMPL Script**

**Parametric Analysis (cont’d)**

**Script run**

```plaintext
ampl: include cutTradeoff.run

Min 20 rolls with waste 62.04

<table>
<thead>
<tr>
<th>Over</th>
<th>Waste</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>46.72</td>
<td>22</td>
</tr>
<tr>
<td>7</td>
<td>47.89</td>
<td>21</td>
</tr>
<tr>
<td>5</td>
<td>54.76</td>
<td>20</td>
</tr>
</tbody>
</table>

ampl:
```
Modeling Language APIs
(Application Programming Interfaces)

Example: Roll Cutting with Pattern Generation
  - fill orders for rolls of various widths

Given
  - Demands, raw width, orders, overrun limit at before
  - pattern generation software
  - result reporting software

Build optimization into an integrated application
  - use AMPL for model-based optimization
  - use a general-purpose programming language for overall control, pattern generation, and reporting
AMPL API

AMPL APIs

Principles

- APIs for “all” popular languages
  - C++, C#, Java, MATLAB, Python, R
- Common overall design
- Common implementation core in C++
- Customizations for each language and its data structures

Key to examples: Python and R

- AMPL entities
- AMPL API Python/R objects
- AMPL API Python/R methods
- Python/R functions etc.
**AMPL Model File**

*Same pattern-cutting model*

```
param nPatterns integer > 0;
set PATTERNS = 1..nPatterns;  # patterns
set WIDTHS;                   # finished widths
param order {WIDTHS} >= 0;    # rolls of width j ordered
param overrun;                # permitted overrun on any width
param rawWidth;                    # width of raw rolls to be cut
param rolls {WIDTHS,PATTERNS} >= 0, default 0;    # rolls of width i in pattern j

var Cut {PATTERNS} integer >= 0;   # raw rolls to cut in each pattern

minimize TotalRawRolls: sum {p in PATTERNS} Cut[p];

subject to FinishedRollLimits {w in WIDTHS}:
    order[w] <= sum {p in PATTERNS} rolls[w,p] * Cut[p] <= order[w] + overrun;
```
**AMPL API**

Some Python Data

*A float, an integer, and a dictionary*

```python
roll_width = 64.5
overrun = 6
Orders = {
    6.77: 10,
    7.56: 40,
    17.46: 33,
    18.76: 10
}
```

*... can also work with lists and Pandas dataframes*
Some R Data

A float, an integer, and a dataframe

```r
roll_width <- 64.5
overrun <- 6
orders <- data.frame(
    width = c( 6.77, 7.56, 17.46, 18.76 ),
    demand = c( 10, 40, 33, 10 )
)
```
**AMPL API**

**Pattern Enumeration in Python**

*Load & generate data, set up AMPL model*

```python
def cuttingEnum(dataset):
    from amplpy import AMPL

    # Read orders, roll_width, overrun
    exec(open(dataset+'.py').read(), globals())

    # Enumerate patterns
    widths = list(sorted(orders.keys(), reverse=True))
    patmat = patternEnum(roll_width, widths)

    # Set up model
    ampl = AMPL()
    ampl.option['ampl_include'] = 'models'
    ampl.read('cut.mod')
```
**Pattern Enumeration in R**

**Load & generate data, set up AMPL model**

```r
cuttingEnum <- function(dataset) {
  library(rAMPL)

  # Read orders, roll_width, overrun
  source(paste(dataset, ".R", sep=""))

  # Enumerate patterns
  patmat <- patternEnum(roll_width, orders$width)
  cat(sprintf("\n%d patterns enumerated\n\n", ncol(patmat)))

  # Set up model
  ampl <- new(AMPL)
  ampl$setOption("ampl_include", "models")
  ampl$read("cut.mod")
}
```
Pattern Enumeration in Python

Send data to AMPL

```python
# Send scalar values
ampl.param['nPatterns'] = len(patmat)
ampl.param['overrun'] = overrun
ampl.param['rawWidth'] = roll_width

# Send order vector
ampl.set['WIDTHS'] = widths
ampl.param['order'] = orders

# Send pattern matrix
ampl.param['rolls'] = {
    (widths[i], 1+p): patmat[p][i]
    for i in range(len(widths))
    for p in range(len(patmat))
}
```
**AMPL API**

**Pattern Enumeration in R**

**Send data to AMPL**

```r
# Send scalar values
ampl$getParameter("nPatterns")$set(ncol(patmat))
ampl$getParameter("overrun")$set(overrun)
ampl$getParameter("rawWidth")$set(roll_width)

# Send order vector
ampl$getSet("WIDTHS")$setValues(orders$width)
ampl$getParameter("order")$setValues(orders$demand)

# Send pattern matrix
df <- as.data.frame(as.table(patmat))
df[,1] <- orders$width[df[,1]]
df[,2] <- as.numeric(df[,2])
ampl$getParameter("rolls")$setValues(df)
```
Pattern Enumeration in Python

Solve and get results

```python
# Solve
ampl.option['solver'] = 'gurobi'
ampl.solve()

# Retrieve solution
CuttingPlan = ampl.var['Cut'].getValues()
cutvec = list(CuttingPlan.getColumn('Cut.val'))
```
**AMPL API**

**Pattern Enumeration in R**

**Solve and get results**

```r
# Solve
ampl$setOption("solver", "gurobi")
ampl$solve()

# Retrieve solution
CuttingPlan <- ampl$getVariable("Cut")$getValues()
solution <- CuttingPlan[CuttingPlan[,-1] != 0,]
```
## Pattern Enumeration in Python

### Display solution

```python
# Prepare solution data

summary = {
    'Data': dataset,
    'Obj': int(ampl.obj['TotalRawRolls'].value()),
    'Waste': ampl.getValue(
        'sum {p in PATTERNS} Cut[p] * 
         (rawWidth - sum {w in WIDTHS} w*rolls[w,p])'
    )
}

solution = [
    (patmat[p], cutvec[p])
    for p in range(len(patmat))
    if cutvec[p] > 0
]

# Create plot of solution

cuttingPlot(roll_width, widths, summary, solution)
```
Pattern Enumeration in R

Display solution

```r
# Prepare solution data
data <- dataset
obj <- ampl$getObjective("TotalRawRolls")$value()
waste <- ampl$getValue(
  "sum {p in PATTERNS} Cut[p] * (rawWidth - sum {w in WIDTHS} w*rolls[w,p])"
)
summary <- list(data=dataset, obj=obj, waste=waste)

# Create plot of solution
cuttingPlot(roll_width, orders$width, patmat, summary, solution)
```
**Pattern Enumeration in Python**

*Enumeration routine*

```python
def patternEnum(roll_width, widths, prefix=[]):
    from math import floor
    max_rep = int(floor(roll_width/widths[0]))
    if len(widths) == 1:
        patmat = [prefix+[max_rep]]
    else:
        patmat = []
        for n in reversed(range(max_rep+1)):
            patmat += patternEnum(roll_width-n*widths[0], widths[1:], prefix+[n])
    return patmat
```
**AMP API**

**Pattern Enumeration in R**

**Enumeration routine**

```r
patternEnum <- function(roll_width, widths, prefix = c()) {
  cur_width <- widths[length(prefix) + 1]
  max_rep <- floor(roll_width / cur_width)
  if (length(prefix) + 1 == length(widths)) {
    return (c(prefix, max_rep))
  } else {
    patterns <- matrix(nrow=length(widths), ncol=0)
    for (n in 0:max_rep) {
      patterns <- cbind(
        patterns,
        patternEnum(roll_width - n * cur_width, widths, c(prefix, n))
      )
    }
    return (patterns)
  }
}
```
**Pattern Enumeration in Python**

**Plotting routine**

```python
def cuttingPlot(roll_width, widths, summ, solution):
    import numpy as np
    import matplotlib.pyplot as plt
    ind = np.arange(len(solution))
    acc = [0]*len(solution)
    colorlist = ['red','lightblue','orange','lightgreen','brown','fuchsia','silver','goldenrod']
```
Pattern Enumeration in R

Plotting routine

```r
function (roll_width, widths, patmat, summary, solution) {
  pal <- rainbow(length(widths))
  par(mar=c(1,1,1,1))
  par(mfrow=c(1,nrow(solution)))
  for (i in 1:nrow(solution)) {
    pattern <- patmat[, solution[i, 1]]
    data <- c()
    color <- c()
  }
}
```
for p, (patt, rep) in enumerate(solution):
    for i in range(len(widths)):
        for j in range(patt[i]):
            vec = [0] * len(solution)
            vec[p] = widths[i]
            plt.barh(ind, vec, 0.6, acc,
                     color=colorlist[i % len(colorlist)], edgecolor='black')
            acc[p] += widths[i]
plt.title(summ['Data'] + ": " +
          str(summ['Obj']) + " rolls" + ", " +
          str(round(100 * summ['Waste'] / (roll_width * summ['Obj']), 2)) + "% waste"
plt.xlim(0, roll_width)
plt.xticks(np.arange(0, roll_width, 10))
plt.yticks(ind, tuple("x {:}".format(rep) for patt, rep in solution))
plt.show()
AMPL API

Pattern Enumeration in R

Plotting routine (cont’d)

```r
for(j in 1:length(pattern)) {
    if(pattern[j] >= 1) {
        for(k in 1:pattern[j]) {
            data <- rbind(data, widths[j])
            color <- c(color, pal[j])
        }
    }
}

label <- sprintf("x %d", solution[i, -1])

barplot(data, main=label, col=color,
        border="white", space=0.04, axes=FALSE, ylim=c(0, roll_width))

print(summary)
```
**AMPL API**

**Pattern Enumeration in Python**

```python
# python

Python 3.4.2 (v3.4.2:ab2c023a9432, Oct 6 2014, 22:16:31) [MSC v.1600 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 6.1.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from pattern Enumeration import *

In [2]: cuttingEnum('Sorrentino')

Gurobi 7.5.0: optimal solution; objective 18
9 simplex iterations
1 branch-and-cut nodes
```

![Figure 1](image-url)
AMPL API

Pattern Enumeration in R

```r
> source("PatternEnumeration.R")
> cuttingEnum("Sorrentino")

95 patterns enumerated

$data
[1] "Sorrentino"

$obj
[1] 18

$waste
[1] 27.12

> |
```

![Graph showing pattern enumeration results](image-url)
Modeling Language Extensions using Programming Languages

Example: *Embedded Python for AMPL (a preview)*

**Sending Python data to an AMPL model**
- via AMPL API for Python
- via Python references in the AMPL model

**Executing Python statements inside AMPL**
- Generate specialized constraints for lot sizing

**Handling callbacks**
- Write callback function in Python
- Export problem + callback, solve, import results
**Embedded Python**

**AMPL Model**

*Symbolic sets, parameters, variables, objective, constraints*

```plaintext
# DATA
set FOOD;
set NUTR;

param cost {FOOD} > 0;
param f_min {FOOD} >= 0;
param f_max {j in FOOD} >= f_min[j];
param n_min {NUTR} >= 0;
param n_max {i in NUTR} >= n_min[i];
param amt {NUTR,FOOD} >= 0;

# MODEL
var Buy {j in FOOD} >= f_min[j], <= f_max[j];
minimize Total_Cost:
    sum {j in FOOD} cost[j] * Buy[j];
subject to Diet {i in NUTR}:
    n_min[i] <= sum {j in FOOD} amt[i,j] * Buy[j] <= n_max[i];
```

*diet.mod*
Embedded Python

Python Data

Lists, dictionaries

```python
food = ['BEF', 'CHK', 'FISH', 'HAM', 'MCH', 'MTL', 'SPG', 'TUR']
cost = {
    'HAM': 2.89, 'BEF': 3.59, 'MCH': 1.89, 'FISH': 2.29,
    'CHK': 2.59, 'MTL': 1.99, 'TUR': 2.49, 'SPG': 1.99
}

amt = [
    [60, 8, 8, 40, 15, 70, 25, 60],
    [20, 0, 10, 40, 35, 30, 50, 20],
    [10, 20, 15, 35, 15, 15, 25, 15],
    [15, 20, 10, 10, 15, 15, 15, 10],
    [928, 2180, 945, 278, 1182, 896, 1329, 1397],
    [295, 770, 440, 430, 315, 400, 379, 450]
]```
Sending Data to AMPL (API)

Call `ampl` methods to read model, send data

```python
from amplpy import AMPL

ampl = AMPL()
ampl.read('diet.mod')

ampl.set['FOOD'] = food
ampl.param['cost'] = cost
ampl.param['f_min'] = f_min
ampl.param['f_max'] = f_max
ampl.set['NUTR'] = nutr
ampl.param['n_min'] = n_min
ampl.param['n_max'] = n_max

ampl.param['amt'] = {
    (n, f): amt[i][j]
    for i, n in enumerate(nutr)
    for j, f in enumerate(food)
}

ampl.solve()
```
Embedded Python

Sending Data to AMPL (Embedded)

Move data correspondences into the model

```plaintext
# SYMBOLIC DATA WITH PYTHON LINKS

$SET[FOOD] { food }

$PARAM[cost{^FOOD}] { cost }

$PARAM[f_min{^FOOD}] { f_min }

$PARAM[f_max{^FOOD}] { f_max }

$SET[NUTR] { nutr }

$PARAM[n_min{^NUTR}] { n_min }

$PARAM[n_max{^NUTR}] { n_max }

$PARAM[amt] {{
    (n, f): amt[i][j]
    for i, n in enumerate(nutr)
    for j, f in enumerate(food)
}}

# MODEL

var Buy { j in FOOD } >= f_min [ j ], <= f_max [ j ];

........
```
Embedded Python

Sending Data to AMPL (Embedded)

Process with PyMPL language extension

```python
from amplpy import AMPL
from pympl import PyMPL

ampl = AMPL(langext=PyMPL())
ampl.read('dietpy.mod')
ampl.solve()
```
Embeded Python

Executing Python inside AMPL

Fix AMPL variables according to Python variable

```AMPL
$PARAM[NT]{8};

var x {1..NT}, >= 0;  # production lot size
var y {1..NT}, binary; # production set-up
var s {0..NT}, >= 0;  # inventory level
var r {1..NT}, ${">= 0" if BACKLOG else ">= 0, <= 0"}$;

# use these variables iff BACKLOG > 0
```

lotsize.mod
Embedded Python

Executing Python inside AMPL

Invoke Python generators for special lot-sizing constraints

```
$EXEC{
    def mrange(a, b):
        return range(a, b+1)
    s = ['s[{}]'.format(t) for t in mrange(0, NT)]
    y = ['y[{}]'.format(t) for t in mrange(1, NT)]
    d = [demand[t] for t in mrange(1, NT)]
    if BACKLOG is False:
        WW_U_AMPL(s, y, d, NT, prefix='w')
    else:
        r = ['r[{}]'.format(t) for t in mrange(1, NT)]
        WW_U_B_AMPL(s, r, y, d, NT, prefix='w')
};
```

```
ampl = AMPL(langext=PyMPL())
ampl.read('lotsize.mod')
ampl.solve()
```
**Embedded Python**

**Executing Python inside AMPL**

**Optional listing of generated constraints**

```plaintext
var ws {wi in 0..8} = s[wi];
var wr {wi in 1..8} = r[wi];
var wy {wi in 1..8} = y[wi];

param wD {1..8, 1..8};
data;
param wD :=
;
model;
```
Executing Python inside AMPL

Optional listing of generated constraints (cont’d)

```plaintext
var wa {1..8};
var wb {1..8};

subject to wXY {wt in 1..8}: wa[wt] + wb[wt] + wy[wt] >= 1;

subject to wXA {wk in 1..8, wt in wk..min(8, wk+8-1): wD[wt,wt]>0}:
    ws[wk-1] >=
        sum {wi in wk..wt} wD[wi,wi] * wa[wi]
        - sum {wi in wk..wt-1} wD[wi+1,wt] * wy[wi];

subject to wXB {wk in 1..8, wt in max(1, wk-8+1)..wk: wD[wt,wt]>0}:
    wr[wk] >=
        sum {wi in wt..wk} wD[wi,wi] * wb[wi]
        - sum {wi in wt+1..wk} wD[wt,wi-1] * wy[wi];
```
Embedded Python

Callbacks

AMPL model with embedded Python

```AMPL
$SET[OBJECTS]{{list(range(n))}};
$SET[RESOURCES]{{list(range(m))}};
$PARAM[value]{{value, i0=0}};
$PARAM[weight]{{
  (i, j): weight[i][j]
  for i in range(n)
  for j in range(m)
}};
$PARAM[capacity]{{capacity, i0=0}};
var x {OBJECTS} >= 0 <= 1 integer;
subject to Limits {r in RESOURCES}:
  sum {i in OBJECTS} weight[i, r] * x[i] <= capacity[r];
maximize Profit:
  sum {i in OBJECTS} value[i] * x[i];
```
Callbacks

Callback function

```python
def callback(model, where):
    global solinfo
    if where == gpy.GRB.Callback.MIPSOL:  # new MIP solution found
        nodecnt = model.cbGet(gpy.GRB.Callback.MIPSOL_NODCNT)
        obj = model.cbGet(gpy.GRB.Callback.MIPSOL_OBJ)
        solinfo.append((nodecnt, obj))  # append to solution list
    solcnt = model.cbGet(gpy.GRB.Callback.MIPSOL_SOLCNT)
    print(
        '** New solution at node {:.0f}, obj {:g}, sol {:d} **'.format(
            nodecnt, obj, solcnt
        ),
        file=log  # write to log.txt
    )
    if time()-t0 >= 10 and solcnt >= 2:
        model.terminate()  # stop solution process and return
```

Embedded Python
Embedded Python

Callbacks

AMPL Python API: Export problem, solve, import solution

```python
from pympl import PyMPL
from amplpy import AMPL
import gurobipy as gpy

ampl = AMPL(langext=PyMPL())
ampl.read('multiknapsack.mod')

grb_model = ampl.exportGurobiModel()
grb_model.params.threads = 1
grb_model.params.timelimit = 10

t0 = time()
solinfo = []  # list to store objective values and node counts
log = open('log.txt', 'w')
grb_model.optimize(callback)

ampl.importGurobiSolution(grb_model)
ampl.display('{i in OBJECTS: x[i] != 0} x[i]')
print(solinfo)  # print stored objective values and node counts
```