AMPL
A Modeling Language for Mathematical Programming

Claudia D'Ambrosio

CNRS researcher
LIX, École Polytechnique, France

STOR-i masterclass — 22 February 2019

Introduction

- Algebraic **modeling language** for linear and nonlinear optimization problems, in discrete or continuous variables.
- Allow the **development** of **models and algorithms**.
- **Linked** to the most widely used **solvers** for LP/MILP/MINLP programming.
- Student version **download**: `ampl.com/products/ampl/ampl-for-students/`
- **AMPL book**: `www.ampl.com/BOOK/download.html`

AMPL files

- Each problem instance is coded in AMPL using three files:
  - a **model file** (extension .mod): contains the mathematical formulation of the problem.
  - a **data file** (extension .dat): contains the numerical values of the problem parameters.
  - a **run file** (extension .run): specifies the solution algorithm (external and/or coded by the user in the AMPL language itself).

The Simplest Example: Nonlinear Knapsack Problem

- We are given N objects and a container with capacity C.
- Every object j has a weight w_j, a profit p_j, and a max available quantity U_j $(j = 1, \cdots, N)$.
- Find for each object the quantity that has to be loaded in the container so as the total weight is not greater than the capacity, and the total profit is maximized.

The Simplest Example: Nonlinear Knapsack Problem

- Variables: quantities x.
- Every variable x_j must be ≥ 0 and $\leq U_j$.
- The profit is a nonlinear function of the the quantity.
- The total weight should be less or equal than the capacity C.
- Maximization of the total profit.
- MINLP problem.

The Simplest Example: Nonlinear Knapsack Problem

$$\max \sum_{j=1}^{N} f_j(x_j)$$

$$\sum_{j=1}^{N} w_j x_j \leq C$$

$$0 \leq x_j \leq U_j \qquad\qquad j = 1, \cdots, N.$$

where $f_j(x_j)$ is any twice continuously differentiable and univariate function $\forall j = 1, \cdots, N$ (e.g., $\frac{c_j}{(1+b_j \exp(-a_j(x_j+d_j)))}$).

The Simplest Example: Nonlinear Knapsack Problem

If a subset J of object can assume only integer value:

$$x_j \text{ integer} \qquad\qquad j \in J \subseteq \{1, \cdots, N\}.$$

Real-world interpretation:
Given a budget C we want to decide how to invest our money so that our profit is maximized. We can buy by choosing among N items, each of which is available for a maximum quantity U_j and each unit of item j costs w_j. Some of the items have to be bought in lots.

Nonlinear Knapsack problem: Citations

1. C. D'Ambrosio, S. Martello. *Heuristic algorithms for the general nonlinear separable knapsack problems*, Computers and Operations Research, 38 (2), pp. 505–513, 2011.
2. H. Kellerer, U. Pferschy, D. Pisinger. *Knapsack Problems*. Springer (2004).
3. S. Martello, P. Toth. *Knapsack problems: Algorithms and computer interpretations*. Wiley-Interscience (1990).

File .run

Typical form of file myFile.run:

```
model myModel.mod;
data myData.dat;
option solver mySolver;
solve;
```

How to call AMPL from the command line:

```
ampl myFile.run
```

or

```
ampl myFile.run > myOutputFile.out
```

Non linear knapsack problem

File .mod:

```
# Author:  Claudia D'Ambrosio
# Date:  20190111
# nlkp.mod


param N > 0;                        # Number of objects
set VARS ordered := 1..N;
param U {j in VARS} > 0, default 100; # Upper bound on object availability
param a {j in VARS} > 0;
param b {j in VARS} > 0;
param c {j in VARS} > 0;
param d {j in VARS} < 0;
param C > 0;                       # Knapsack capacity


var x {j in VARS} >= 0, <= U[j]; # defining variables, their bounds


maximize Total_Profit:
   sum {j in VARS} (a[j]+b[j]*x[j]+c[j]*x[j]**2+d[j]*x[j]**3);


subject to KP_constraint:
   sum{j in VARS} x[j] <= C;
```

Non linear knapsack problem

File .dat:

```
# Author:  Claudia D'Ambrosio
# Date:  20190121
# nlkp.dat

param N := 10;
param C := 546.000000;

param:  a :=
1 0.172274
2 0.134944
3 0.101030
4 0.163588
5 0.152350
6 0.196601
7 0.181208
8 0.126588
9 0.184087
10 0.187434
;
```

```
param:  b :=
1 78.770199
2 77.468892
3 93.324757
4 96.180080
5 55.137398
6 40.101851
7 36.007819
8 5.317250
9 9.964929
10 60.265707
;

param:  c :=
1 3.062328
2 43.280130
3 52.983122
4 62.101010
5 58.531125
6 47.574366
7 53.101406
8 6.902601
9 16.985577
10 62.576610
;
```

```
param:  d :=
1 -81.876165
2 -56.455229
3 -56.428945
4 -43.813029
5 -28.246895
6 -81.108142
7 -37.839956
8 -1.138258
9 -80.968161
10 -11.536015
;

param:  U :=
1 100.000000
2 100.000000
3 100.000000
4 100.000000
5 100.000000
6 100.000000
7 100.000000
8 100.000000
9 100.000000
10 100.000000
;
```

Non linear knapsack problem

File .run:

```
# Author:  Claudia D'Ambrosio
# Date:  20190121
# nlkp.run


reset;
reset data;


model nlkp.mod;


data "/mypath/nlkp.dat";


option solver "/usr/local/bin/baron";


option baron_options 'prfreq=100 outlev=1';


solve > nlkp.out;
```

Non linear knapsack problem

File nlkp.out obtained:

```
BARON 18.11.12 (2018.11.12):  prfreq=100 outlev=1
================================================================================
...
================================================================================
Preprocessing found feasible solution with value 1.60010400000
Doing local search
Preprocessing found feasible solution with value 1093.96317285
Solving bounding LP
Starting multi-start local search
Done with local search
================================================================================
Iteration Open nodes Time (s) Lower bound Upper bound
1 1 0.15 1093.96 11340.2
100 16 1.62 1093.96 1121.50
200 30 2.04 1093.96 1094.05
300 35 2.19 1093.96 1094.00
400 31 2.38 1093.96 1093.97
500 28 2.55 1093.96 1093.97
600 13 2.67 1093.96 1093.97
631 0 2.70 1093.96 1093.96
Cleaning up
*** Normal completion ***
Wall clock time:  3.00
Total CPU time used:  2.70
Total no.  of BaR iterations:  631
Best solution found at node:  -1
Max.  no.  of nodes in memory:  37
All done
================================================================================
BARON 18.11.12 (2018.11.12):  631 iterations, optimal within tolerances.
Objective 1093.963173

Profit = 1093.96
```

To install AMPL

- Download one of the following .zip files (no IDE, run command-line executable):
  - `http://www.lix.polytechnique.fr/~dambrosio/teaching/MPRO/PMA-2019/ampl_linux-intel32.zip`
  - `http://www.lix.polytechnique.fr/~dambrosio/teaching/MPRO/PMA-2019/ampl_linux-intel64.zip`
  - `http://www.lix.polytechnique.fr/~dambrosio/teaching/MPRO/PMA-2019/ampl_macosx64.zip`
  - `http://www.lix.polytechnique.fr/~dambrosio/teaching/MPRO/PMA-2019/ampl_mswin32.zip`
  - `http://www.lix.polytechnique.fr/~dambrosio/teaching/MPRO/PMA-2019/ampl_mswin64.zip`
- Follow installation instructions: `https://ampl.com/try-ampl/ampl-for-courses/ampl-course-install/`
- Available solvers: baron, conopt, cplex, gurobi, ilogcp, knitro, lgo, loqo, minos, snopt, xpress

References

- Modeling languages like **ampl**: `ampl.com`
  or **gams**: `www.gams.com`
- Open source solvers like coin-or **ipopt/bonmin/couenne**:
  `www.coin-or.org/projects/`
  and **scip**: `scip.zib.de`
- **NEOS Server**, State-of-the-Art Solvers for Numerical Optimization: `www.neos-server.org/neos/`
- **MINLP benchmarks** like `minlp.org`
  `www.gamsworld.org/minlp/minlplib2/html/`
  `wiki.mcs.anl.gov/leyffer/index.php/MacMINLP`

Exercises

1. Implement the knapsack problem model by considering the profit of each item j equal to $a_j + b_j x_j + c_j x_j^2 + d_j x_j^3$ and solve the instance nlkp.dat (you can download it from the course web page) with KNITRO and with BARON.
2. Are the optimal solutions found by the two solvers equivalent?
3. Considering that $a, b, c > 0$ and $d < 0$, is the problem convex? Why?
4. Implement the knapsack problem model by considering the profit of each item j equal to $d_j + c_j x_j + b_j x_j^2 + a_j x_j^3$ and solve the instance nlkp.dat with KNITRO and with BARON. Answer to questions 2. and 3.
5. Now use $c[j]/(1 + b[j] * exp(-a[j] * (x[j] + d[j])))$ as profit function for item j and solve the instance nlkp.dat with KNITRO and with BARON. Answer to questions 2. and 3.

Exercises

1. Implement multi-start algorithm for NLKP and Snopt as solver (AMPL commands needed: `for`, `let`, `Uniform`).
2. Implement the pooling problem model, the haverly instance, and solve it through Baron.
3. Implement the relaxation of the pooling problem (see McCormick), the haverly instance, and solve it through Cplex. What's the difference between the obtained solution and the solution of the previous point?
4. As for point 1., but with binary variables by adding a fixed cost of 50 for arc (1,1) and 55 for (1,2).
5. As for point 2., but with binary variables.
6. Model in AMPL the optimization problem described in "Optimizing the Design of Water Distribution Networks Using Mathematical Optimization" (see webpage) and solve the shamir.dat instance (available on the webpage).

Haverly instance

- Sets cardinality: $|I| = 3$, $|L| = 2$, $|J| = 2$, $|K| = 1$.
- Arcs set: $\{(i_1, l_1), (i_2, l_1), (i_3, o_1), (i_3, o_2), (l_1, o_1), (l_1, o_2)\}$.
- Cost $c = (6, 16, 10)$.
- Profit $p = (9, 15)$.
- $\lambda = (3, 1, 2)$.
- $\alpha = (2.5, 1.5)$.
- No arc capacity.
- No node capacity for I and L. For outputs, capacity $= (100, 200)$.