

# Constraint Programming in AMPL

Victor Zverovich



**AMPL Optimization**

CP Solvers: Modeling, Applications, Integration, and Standardization  
CP2013, Uppsala, Sweden, September 16, 2013

# About AMPL


- **AMPL** is a popular modeling system
  - used in businesses, government agencies, and academic institutions (over 100 courses in 2012)
  - large community (> 1,400 members in **AMPL Google Group** alone)
  - the most popular input format on **NEOS** (> 200,000 or 57% submissions in 2012)
- AMPL is high-level, solver-independent and efficient.
- Supports a variety of solvers and problem types: linear, mixed integer, quadratic, second-order cone, nonlinear, complementarity problems and more.

# AMPL Application Areas

- Transportation (air, rail, truck)
- Production planning and supply chain
- Finance (investment banking, insurance)
- Natural resources (electric power, gas, mining)
- Telecommunications
- Internet services

Examples:

**ZARA** - clothing and accessories retailer

 **Norske Skog** - paper manufacturer

# Solver Support

- AMPL provides consistent interface to a large number of solvers.
- Switching between solvers is easy.
- Connecting new solvers is easy using the open-source AMPL Solver Library.
- Solver-specific features are supported such as the whole set of solver options.
- Features missing in a solver are often implemented in an AMPL solver driver. Examples: meaningful solution log, reformulation of missing constructs.

# Connected CP Solvers

- Solvers:
  - ilogcp: **IBM/ILOG CP Optimizer**
  - gecode: **Generic constraint development environment.**  
**New: Gecode 4.2**
  - jacop: **Java constraint solver**
- How to get:
  - Ilogcp is available to all CPLEX-for-AMPL users
  - AMPL Gecode and JaCoP (soon) downloads:  
<https://code.google.com/p/ampl/>
  - Source code: [https://github.com/vitaut/ampl\\_solvers/gecode](https://github.com/vitaut/ampl_solvers/gecode) | [solvers/ilogcp](https://github.com/vitaut/ampl_solvers/ilogcp) | [solvers/jacop](https://github.com/vitaut/ampl_solvers/jacop)

# AMPL User Interfaces

- Classical command-line interactive environment
- Eclipse-based IDEs:
  - AMPL IDE
  - AMPLDev (adds stochastic programming features)
- Solver Studio for Excel
- Matlab (TOMLAB)
- IPython (experimental)

# AMPL IDE (Beta)

Based on the Eclipse platform well-known for its Java and Android IDEs.

Some of the features:

- Cross-platform with native look and feel on each platform
- Interactive console with command history
- Context-sensitive syntax highlighting
- Quick links to error locations
- Solution process can be interrupted by a user at any time and the best solution found so far will be available
- Freely available from <http://www.ampl.com/IDE/beta.html>

# AMPL IDE (Beta)

The screenshot displays the AMPL IDE (Beta) interface. The window title is "AMPL IDE" and the menu bar includes "File", "Edit", "Window", and "Help".

**File Explorer:** The left sidebar shows the current directory as `/home/viz/models`. The file list includes folders `compl`, `gsl`, `logic`, `nlmodels`, and `tables`, along with files `openshop.dat` and `openshop.mod`.

**Code Editor:** The main editor window shows the `openshop.mod` file with the following AMPL code:

```
minimize Objective: Makespan;

subject to NoJobConflicts
  {m in 1..nMach, j1 in 1..nJobs, j2 in j1+1..nJobs}:
  Start[m,j1] + duration[m,j1] <= Start[m,j2] or
  Start[m,j2] + duration[m,j2] <= Start[m,j1];

subject to NoMachineConflicts
  {m1 in 1..nMach, m2 in m1+1..nMach, j in 1..nJobs}:
  Start[m1,j] + duration[m1,j] <= Start[m2,j] or
  Start[m2,j] + duration[m2,j] <= Start[m1,j];
```

**Console:** The console window at the bottom shows the execution output:

```
AMPL
ampl: model openshop.mod;
ampl: data openshop.dat;
ampl: option solver gencode;
ampl: solve;
gencode 4.0.0: optimal solution
1784770 nodes, 892382 fails, objective 1955
ampl:
```



# Solver Studio for Excel

- Create and edit AMPL models without leaving Excel
- Solve using local solvers or in the cloud via NEOS
- Integrated model and data editors
- Automatic data exchange with model
- Freely available from <http://solverstudio.org>

# Solver Studio for Excel

The screenshot displays the SolverStudio for AMPL interface within Microsoft Excel. The main window shows a data table with columns for Ingredients, Costs, and various nutrients (Protein, Fat, Fibre, Salt, OneCan), along with an Amount column. The SolverStudio window on the right contains the AMPL model code and the output of the solver.

Ingredients	Costs	Contributes					Amount
		Protein	Fat	Fibre	Salt	OneCan	
Chicken	0.013	0.1	0.08	0.001	0.002	1	0
Beef	0.008	0.2	0.1	0.005	0.005	1	10
Mutton	0.01	0.15	0.11	0.003	0.007	1	0
Rice	0.002	0	0.01	0.1	0.002	1	0
Wheat bran	0.005	0.04	0.01	0.15	0.008	1	0
Gel	0.001	0	0	0	0	1	90

```

set INGREDIENTS;
set REQUIREMENTS;
param Cost {INGREDIENTS};
param Contributes {REQUIREMENTS, INGREDIENTS};
param Lower {REQUIREMENTS};
param Upper {r in REQUIREMENTS} >= Lower[r];
var Amount {INGREDIENTS} >= 0;

minimize TotalCost:
    sum {i in INGREDIENTS} Cost[i] * Amount[i];
subject to MeetRequirements {r in REQUIREMENTS}:
    Lower[r] <= sum {i in INGREDIENTS} Contributes[r,i] * Amount[i];
    Upper[r] >= sum {i in INGREDIENTS} Contributes[r,i] * Amount[i];

data SheetData.dat; # Get data from the spreadsheet;

option solver cplexamp;
solve;

display Amount > Sheet; # Write the solution back to the spreadsheet;
    
```

Model Output

```

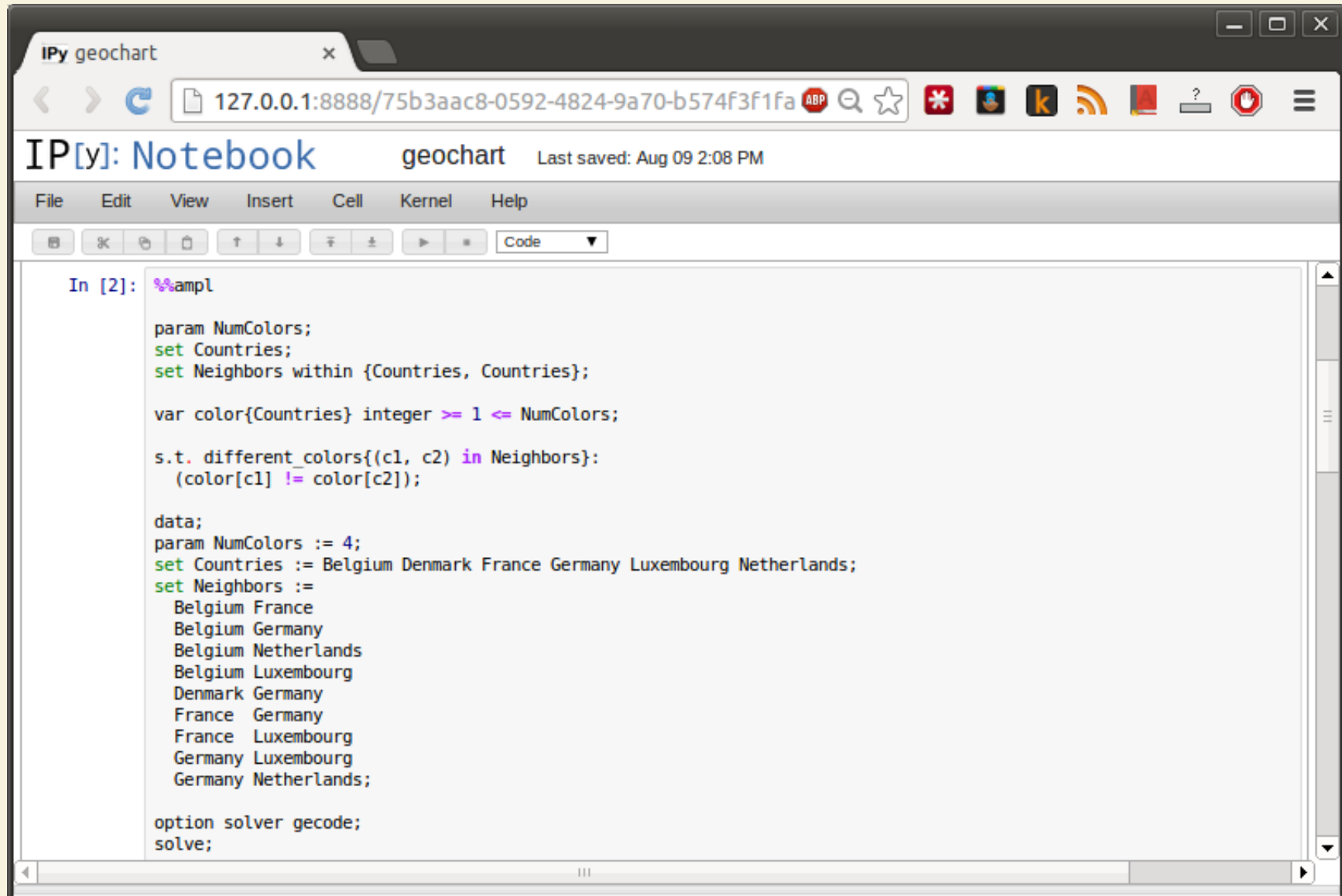
AMPL Version 20080102 (x86_win32)
CPLEX 11.0.0: optimal solution; objective 0.17
2 dual simplex iterations (0 in phase I)

## AMPL run completed.
    
```

Row 19, Column 35

Ready | Average: 16.66666667 | Count: 6 | Min: 0 | Max: 90 | Sum: 100 | 100%

# AMPL IPython Plugin



The image shows a screenshot of a web browser displaying an IPython Notebook. The browser's address bar shows the URL `127.0.0.1:8888/75b3aac8-0592-4824-9a70-b574f3f1fa`. The notebook interface includes a menu bar with options: File, Edit, View, Insert, Cell, Kernel, and Help. Below the menu bar is a toolbar with various icons for file operations and execution. The main content area shows a code cell with the following AMPL code:

```
In [2]: %ampl

param NumColors;
set Countries;
set Neighbors within {Countries, Countries};

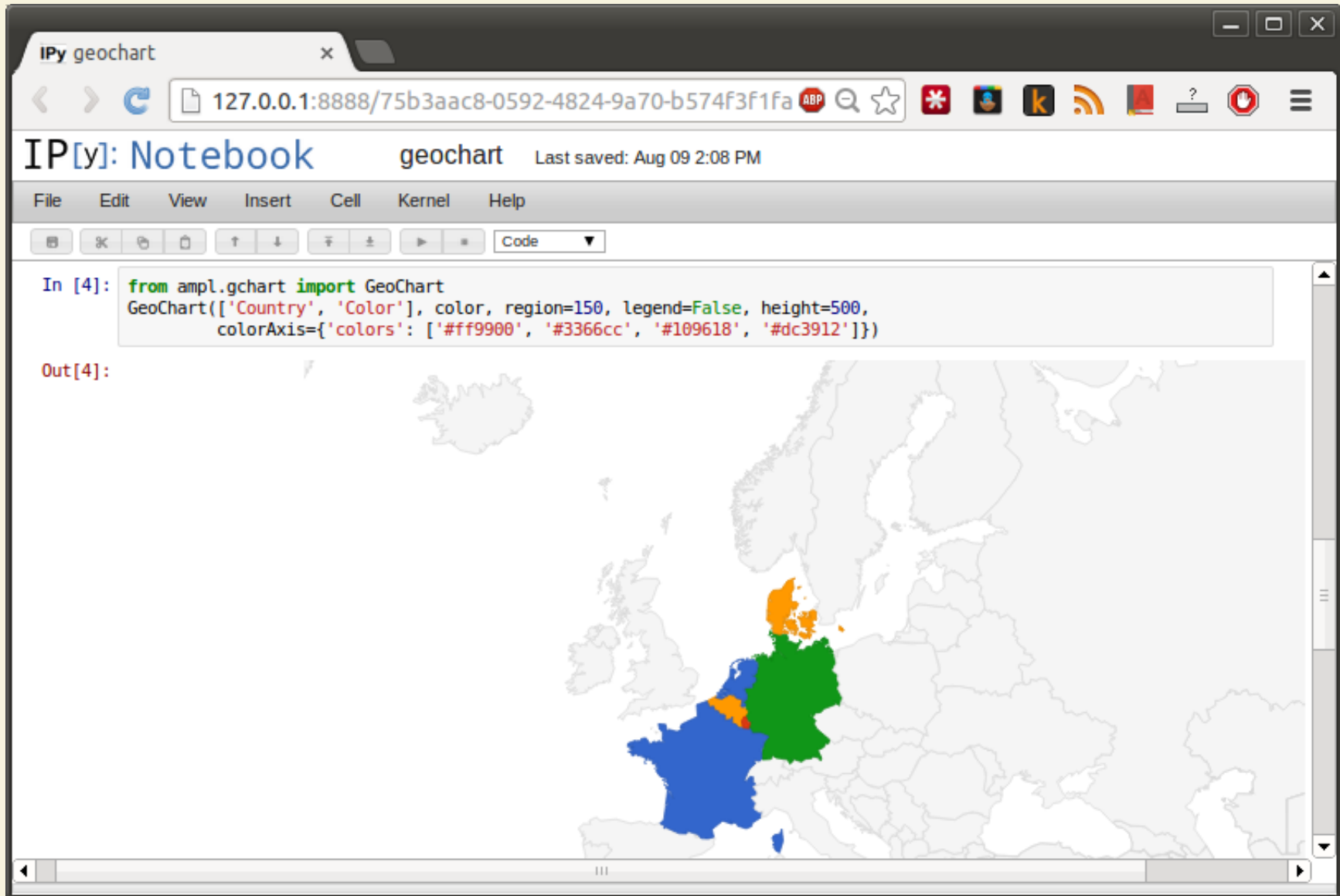
var color{Countries} integer >= 1 <= NumColors;

s.t. different_colors{(c1, c2) in Neighbors}:
    (color[c1] != color[c2]);

data;
param NumColors := 4;
set Countries := Belgium Denmark France Germany Luxembourg Netherlands;
set Neighbors :=
    Belgium France
    Belgium Germany
    Belgium Netherlands
    Belgium Luxembourg
    Denmark Germany
    France Germany
    France Luxembourg
    Germany Luxembourg
    Germany Netherlands;

option solver gecode;
solve;
```

# AMPL IPython Plugin

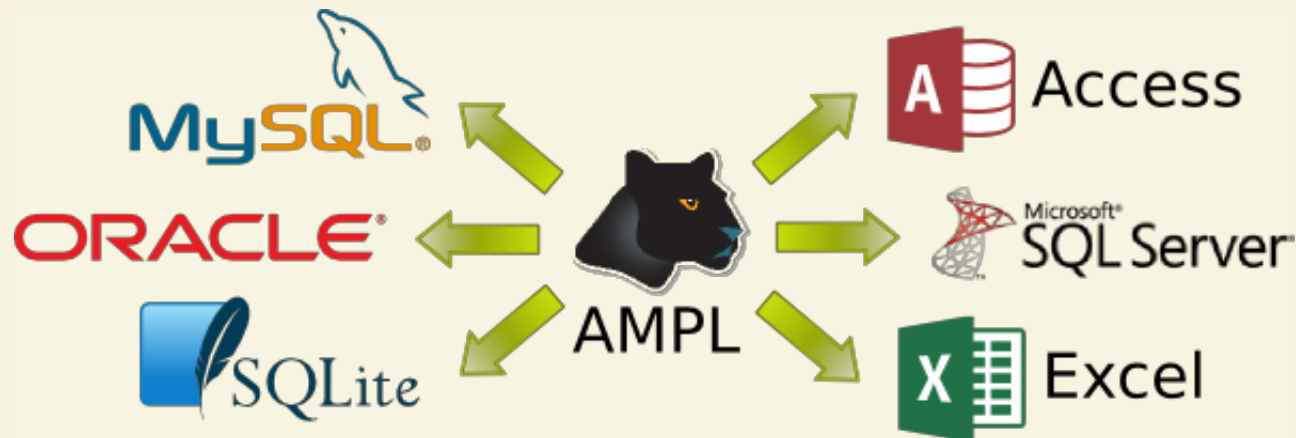


# Database and Spreadsheet Access

- Crucial for integration into real-world applications
- AMPL provides streamlined database access on major platforms



- Supports any database system that has an ODBC driver



# History of CP Support in AMPL

- 1996: first experiments with adding logic programming features to AMPL.
- Fourer (1998). *Extending a General-Purpose Algebraic Modeling Language to Combinatorial Optimization: A Logic Programming Approach* [1]
- Fourer and Gay (2001). *Hooking a Constraint Programming Solver to an Algebraic Modeling Language*
- Fourer and Gay (2002). *Extending an Algebraic Modeling Language to Support Constraint Programming* [2]
- Initially **IBM/ILOG CP Optimizer** was connected.
- **Gecode** was connected in 2012, **JaCoP** - in early 2013.

# Supported CP Constructs

- Logical operators: and, or, not
- Iterated logical operators: exists, forall
- Conditional operators:  
if-then, if-then-else,  $\implies$ ,  $\implies$  else,  $\iff$ ,  $\iff$
- Counting operators:  
count, atmost, atleast, exactly, numberof
- Pairwise operator: alldiff
- All kinds of arithmetic expressions and functions available in AMPL (if can be handled by a solver or reformulated)

See <http://www.ampl.com/NEW/LOGIC/> for details.

# Example: Transportation Model

An example from a multicommodity transportation model  
**multmip3.mod**

For every origin  $i$  and destination  $j$  the total shipments  
 $\sum \{p \text{ in PROD}\} \text{Trans}[i, j, p]$  should be either zero or  
between  $\text{minload}$  and  $\text{limit}[i, j]$ .

MIP formulation:

```
var Trans {ORIG,DEST,PROD} >= 0;
var Use {ORIG,DEST} binary;
subject to Multi {i in ORIG, j in DEST}:
    sum {p in PROD} Trans[i,j,p] <= limit[i,j] * Use[i,j];
subject to Min_Ship {i in ORIG, j in DEST}:
    sum {p in PROD} Trans[i,j,p] >= minload * Use[i,j];
```



# Transportation Example using CP

Disjunctive constraint:

```
subject to Multi_Min_Ship {i in ORIG, j in DEST}:  
  sum {p in PROD} Trans[i,j,p] = 0 or  
  minload <= sum {p in PROD} Trans[i,j,p] <= limit[i,j];
```

Implication:

```
subject to Multi_Min_Ship {i in ORIG, j in DEST}:  
  sum {p in PROD} Trans[i,j,p] > 0 ==>  
  minload <= sum {p in PROD} Trans[i,j,p] <= limit[i,j];
```

- No need for auxiliary binary variables.
- The formulation is more straightforward.

# Example: Scheduling Model

The goal is to find a minimal cost assignment of jobs to machines.

MIP formulation:

```
param n integer > 0;

set JOBS := 1..n;
set MACHINES := 1..n;

param cap {MACHINES} integer >= 0;

param cost {JOBS,MACHINES} > 0;
var Assign {JOBS,MACHINES} binary;

minimize TotalCost:
    sum {j in JOBS, k in MACHINES} cost[j,k] * Assign[j,k];

subj to OneMachinePerJob {j in JOBS}:
    sum {k in MACHINES} Assign[j,k] = 1;

subj to CapacityOfMachine {k in MACHINES}:
    sum {j in JOBS} Assign[j,k] <= cap[k];
```

# Scheduling Example using CP

Using the count operator:

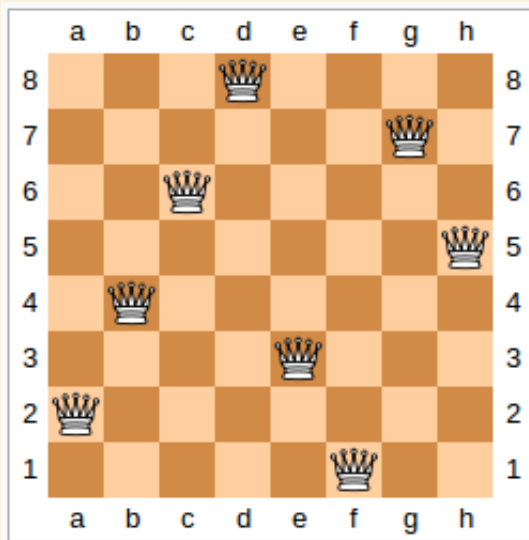
```
subj to CapacityOfMachine {k in MACHINES}:  
    count {j in JOBS} (MachineForJob[j] = k) <= cap[k];
```

Using the numberof operator:

```
subj to CapacityOfMachine {k in MACHINES}:  
    numberof k in ({j in JOBS} MachineForJob[j]) <= cap[k];
```

- No need for  $n^2$  binary variables.
- All the numberof constraints can be converted into a single `IloDistribute` constraint in `ilogcp`.

# N Queens Example with Alldiff



---

```
# Place n queens on an n by n board  
# so that no two queens can attack  
# each other (nqueens.mod).
```

```
param n integer > 0;  
var Row {1..n} integer >= 1 <= n;
```

```
s.t. c1: alldiff ({j in 1..n} Row[j]);  
s.t. c2: alldiff ({j in 1..n} Row[j]+j);  
s.t. c3: alldiff ({j in 1..n} Row[j]-j);
```

---

More examples available at

<http://www.ampl.com/NEW/LOGIC/EXAMPLES>.

# Work in Progress

- Variables in subscripts
- Multiple solutions
- Element constraint
- Restart functionality in the Gecode driver
- Use constraint suffixes (attributes) for fine-grained control over some of the search options, e.g. `icl` in Gecode.

# Summary

- AMPL provides a consistent and intuitive interface to multiple constraint programming solvers.
- CP functionality in AMPL is production-ready and new features are actively added.
- New user interfaces make model development easier.
- Database access functionality facilitates integration into real-world applications.

# Links

- AMPL Logic and Constraint Programming Extensions:  
<http://www.ampl.com/NEW/LOGIC/>
- Trial version of AMPL with IBM/ILOG CP:  
<http://www.ampl.com/trial.html>
- Open-source AMPL solvers and libraries including Gecode:  
<https://code.google.com/p/ampl/>
- AMPL models by Hakan Kjellerstrand including 100 CP models: <http://www.hakank.org/ampl/>
- Source code for ilogcp, gecode and jacop interfaces on GitHub: <https://github.com/vitaut/ampl>