# Basic Numberjack Tutorial

## Adapted from Hebrard et al.'s AAAI 2010 tutorial and parts of the Numberjack website

CS 175

April 5, 2011

# What is Numberjack?

- A platform for constraints
- Written in Python - a front-end to C++-based solvers
- Excellent for rapidly trying out models

# What is Numberjack?

- A platform for constraints
- Written in Python - a front-end to C++-based solvers
- Excellent for rapidly trying out models
- "Cuts your exponential search tree into logs"

# Overview of Python

- Scripting language
- Supports classes, objects, etc.
- Duck-typing

# Variables and Functions

Variables

```
a = 2
```

- No need to declare the variable
- Variables are untyped

Functions

```
def double(a):
    return a * 2
```

- Functions are also not typed
- Indentations based on whitespace and are part of the syntax

# Lists and Tuples

Lists

```
foo = [1, 4, 5, 10, 2]
bar = ["this", "is", "a", "list"]
```

Tuples

```
triplet = (1, 2, 3)
course = ("CS", 175)
```

- Again, types don't matter even within lists and tuples

# Control

```
if <boolean_exp>:
    do_stuff()

while <boolean_exp>:
    do_stuff()
```

# For Loops

For loops in C/C++/Java

```
for (int i = 0; i < n; ++i) {
    do_stuff(i)
}
```

For loops in Python

```
for i in range(n):
    do_stuff(i)
```

# More Fun with For Loops

```
for element in list:
    do_stuff_with(element)

teamProjects = [(0, "Asteroid Simulation"),
                (1, "Scrabble"),
                (2, "Poker")]
for teamNumber, project in teamProjects:
    print "Team", teamNumber, ":", project

Team 0 : Asteroid Simulation
Team 1 : Scrabble
Team 2 : Poker
```

# List Comprehensions

A very useful feature!

```
>>> range(4)
[0, 1, 2, 3]
>>> [x * 2 for x in range(4)]
[0, 2, 4, 6]
>>> [x * 2 for x in range(4) if x >= 2]
[4, 6]
```

Generally,

```
[<expression> for x in <Iterable> (if <condition>)]
```

# Overview

- Constructs
  - Variables
  - Constraints
  - Model
- A common API to interface with back-end solvers

# Variables

```
# binary variable
Variable()
# domain from 0 to N−1
Variable(N)
# domain from L to U
Variable(L, U)
# domain specified by a list
Variable(list)
```

Useful method (used after a solution has been found)

```
get_value()
```

# Variables

More constructors:

*# create a list of N binary variables*
`VarArray(N)`
*# create a list of N variables with domains from 0 to D−1*
`VarArray(N, D)`
*# create a list of N variables with domains from L to U*
`VarArray(N, L, U)`

# Variables

...and even more constructors:

```
# create a matrix of M x N binary variables
m = Matrix(M, N)
# create a matrix of M x N variables with domains from L to U
m = Matrix(M, N, L, U)
```

Special operators

```
# Return a VarArray containing all of the elements of the Matrix
m.flat
# Return a list of VarArrays corresponding to each row
m.row
# Return a list of VarArrays corresponding to each column
m.col
```

# Constraints

- Arithmetic operators on variables

```
x > y
x == y + 2
a[1] > b[2]
m[1][4] != n[4][3]
```

- Global constructors

```
AllDiff([a, b, c, d, e])
AllDiff(myVarArray)
AllDiff(myMatrix)
Sum([a, b, c, d]) >= e
```

# Model

- Used to collect the constraints together to define a problem
- Constructors

```
# empty model
model = Model()

# model with constraints
model = Model(constraints,...)
```

- Adding more constraints

```
model.add(constraints)
#or
model += constraints
```

# Using a Solver

- Different solvers available (Mistral, MiniSat, Walksat)
- Methods

```
# Get a solver to solve the given problem specified
# by the model,
solver = model.load('nameOfSolver')
# attempts to solve the problem
solver.solve()
# for search-based solvers only (to generate multiple solutions)
solver.startNewSearch()
while solver.getNewSolution():
    # do something with solution
```

- Results are stored in the Variable objects

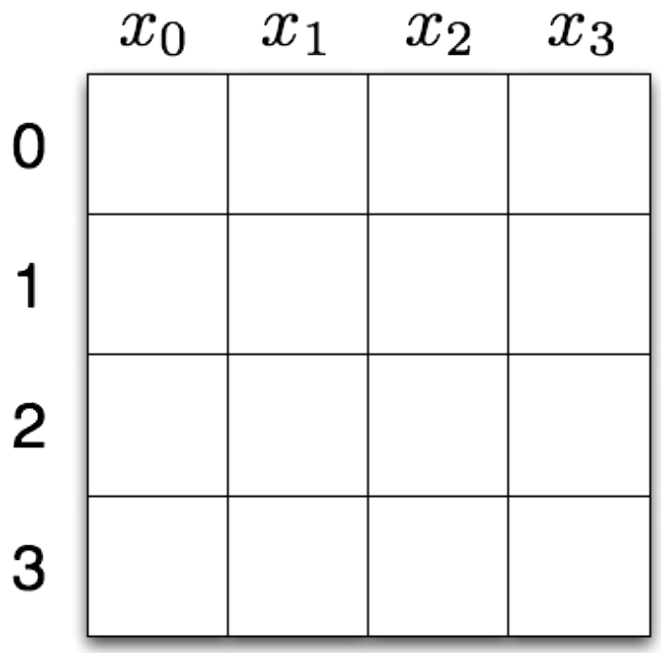
# Outline of Usage

- Specify variables
- Specify constraints over those variables
- Construct a model with the constraints
- Construct the solver using that model
- Call solve() and extract results from Variables using get_value()

# Outline of Usage

- Specify variables
- Specify constraints over those variables
- Construct a model with the constraints
- Construct the solver using that model
- Call solve() and extract results from Variables using get_value()
- Can alternatively use the print statement on Variables directly to output their values

# Problem Definition




- Place queens on the chessboard such that no two queens are attacking each other

# Modeling

- What are the variables/domains of variables for the 4-queens problem presented?

# Modeling

- What are the variables/domains of variables for the 4-queens problem presented?
- How about in general for the N-queens problem?

# Modeling

- What are the variables/domains of variables for the 4-queens problem presented?
- How about in general for the N-queens problem?
- What constraints do we need?

# Problem Definition

| | | | |
|---|---|---|---|
| 1 | 2 | 15 | 16 |
| 12 | 14 | 3 | 5 |
| 13 | 7 | 10 | 4 |
| 8 | 11 | 6 | 9 |

- Given an $N \times N$ square, place numbers ranging from 1 to $N^2$ such that each row, column, and diagonal has the same sum

# Modeling

- Same questions as before...(variables, domains, constraints?)

# Modeling

- Same questions as before...(variables, domains, constraints?)
- Which Numberjack Variable constructor seems appropriate for this?

# Problem Definition

| Items | Bid Amount | Variable |
|---|---|---|
| A, B | 10 | $x_0$ |
| A, C | 20 | $x_1$ |
| B, D | 20 | $x_2$ |
| B, C, D | 25 | $x_3$ |
| A | 14 | $x_4$ |

- Choose bids such that sets of items across bids are disjoint

# Problem Definition

| Items | Bid Amount | Variable |
|---|---|---|
| A, B | 10 | $x_0$ |
| A, C | 20 | $x_1$ |
| B, D | 20 | $x_2$ |
| B, C, D | 25 | $x_3$ |
| A | 14 | $x_4$ |

- Choose bids such that sets of items across bids are disjoint
- ....such that the selection maximizes the revenue

# Problem Definition

| Items | Bid Amount | Variable |
|---|---|---|
| A, B | 10 | $x_0$ |
| A, C | 20 | $x_1$ |
| B, D | 20 | $x_2$ |
| B, C, D | 25 | $x_3$ |
| A | 14 | $x_4$ |

- Choose bids such that sets of items across bids are disjoint
- ....such that the selection maximizes the revenue
- Different from constraint satisfaction...known as constraint optimization
- In addition to constraints, we need to specify an objective function

# Modeling

- The number of variables is given this time, but what are the domains?
- What is the objective function?
- What are the constraints?

# Conclusion

- Rapid prototyping of problems
- Easy to test out different solvers
- Numberjack website: http://numberjack.ucc.ie (also linked from the course page)