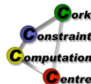
Constraint Programming and Combinatorial Optimisation in Numberjack

Emmanuel Hebrard, Eoin O'Mahony, Barry O'Sullivan

LAAS-CNRS
Cork Constraint Computation Centre, University College Cork
This work was supported by Science Foundation Ireland (Grant No. 05/IN/I886).



December 17, 2010

Outline

1. Introduction
2. Basic Modeling
3. Advanced Modeling and Search
4. Controling Search
5. Behind the Scene

Outline

1. Introduction
2. Basic Modeling
3. Advanced Modeling and Search
4. Controling Search
5. Behind the Scene

Introduction

Numberjack cuts your exponential search tree into logs!

Numberjack

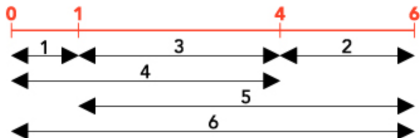
- A Python API for combinatorial optimization
- Why do we need yet another one?

Combinatorial Optimization

- A (broad) class of problems:
  - Make a number of decisions (Variables $\mapsto$ Domains)
  - Subject to a set of constraints (Constraints)
  - In order to optimize an objective
- A (broad) class of methods - Search and/or Inference
  - Constraint Programming (CP)
  - Boolean Satisfiability (SAT)
  - Mixed Integer Programming (MIP)
  - Local Search / Metaheuristics
- Scheduling, Timetabling, Configuration, Design, Diagnosis, Mathematics, Frequency assignment, Bioinformatics,...

Modeling - the constraint programming view

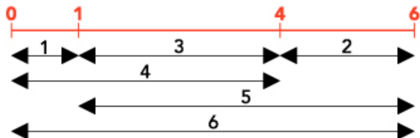
An Example: Golomb Ruler



- Place N marks on a ruler
- Not twice the same distance
- Minimise the length of the ruler

Modeling - the constraint programming view

An Example: Golomb Ruler



- Place N marks on a ruler
- Not twice the same distance
- Minimise the length of the ruler

Variables, Domains, Constraints and Objective

- Variables & Domains: The marks & their positions on the ruler: $m_1, m_2, m_3, m_4 \in [0, 6]$.
- Constraints: For all marks i, j, k, l: $(m_i - m_j) \neq (m_k - m_l)$.
- Objective: Minimise $last - first$.

Python

Python's Features

- High level & interpreted
- Duck typed
  - If it looks like a duck and quacks like a duck its a duck!
- Object oriented (or not!)
- Extensible
  - Modules in C/C++
- Large community

The basics

Data structures

- **list**

```
primes = [2, 3, 5, 7, 11]
cities = ["Cork", "Toulouse", "Ithaca"]
print cities[2]
>>> "Ithaca"

print cities[1:]
>>> ["Toulouse", "Ithaca"]
```

- **tuple**
- **dict**

The Basics

Loops

```
for item in <iterable>:
   do_something_with(item)
```

```
scores = [("Villa", 5),
          ("Sneijder", 5),
          ("Henry", 0)]

for player, goals in scores:
   print player, " scored ", goals, " goals"
```

The less basics

List comprehensions

One of Python's most useful feature for concise code is list comprehensions. The implementation is similar to that of Haskell.

```
print range(4)
>>> [0, 1, 2, 3]
print [x*x for x in range(4)]
>>> [0, 1, 4, 9]
```

List comprehensions generally take the following form.

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

What is Numberjack?

A platform for combinatorial optimization

- Common language for diverse paradigms
- Back-end solvers are C/C++ libraries
  - CP (Mistral)
  - MIP (SCIP)
  - SAT (MiniSat, Walksat)
- API with a language "feel"
  - Readable, fast development time
  - Full access to the back-end solvers (SWIG)
- Open source (LGPL)
- Work in progress...
  - We need you to get involved.

Outline

1 Introduction

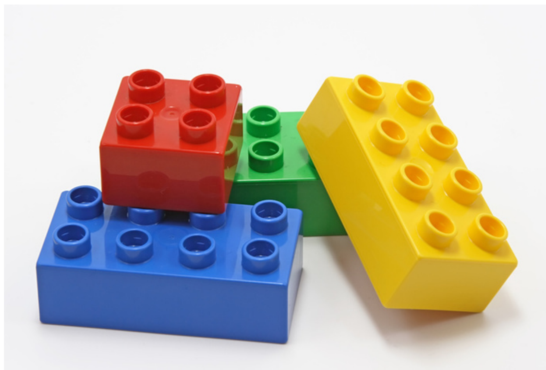
2 Basic Modeling

3 Advanced Modeling and Search

4 Controling Search

5 Behind the Scene

Basic Modeling and Solving



Building Blocks

- Basics of modeling in Numberjack
- Solving your problems

Expression Tree

Module

- Numberjack is a module
- Each solver is a module

```
from Numberjack import *
import MiniSat
```

Model

- Variables
- List of constraints
- Solver-abstracted

```
X = Variable( [2,4,5,8,10] )
model = Model( X != Y, ... )
model.add( Z == 2*x1 + x2 - 3*x3 )
```

Solver

- Build from a model
- Accessors are wrapped

```
solver = MiniSat.Solver( model )
solver.solve()
```

Expression Tree

Variables - Leaves of an expression tree

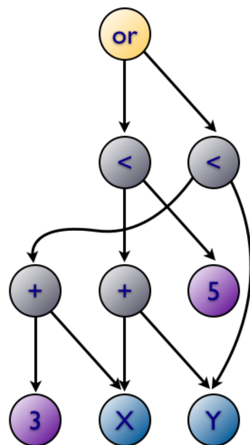
```
X = Variable(0,10)
X = Variable([1,3,5,7])
```

Predicates - Internal nodes of an expression tree

```
P = X+Y          # arythmetic value
Q = X+3 <= Y     # truth (logic) value
```

Constraints - Root node of an expression tree

```
C1 = (X+Y < 5) | (X+3 < Y)
C2 = AllDiff([x,y,z])
C3 = Sum([a,b,c,d]) >= e
```



Demo!

Golomb Ruler in Numberjack

```
m = 8
n = 2**m

marks = [Variable(n) for i in range(m)]

model = Model(

    Minimise( marks[-1] ),

    [marks[i-1] < marks[i] for i in range(1,m)],

    AllDiff( [ (first - second) for first, second in pair_of(marks) ] ),

    marks[0] == 0
)

solver = Mistral.Solver( model, marks )
solver.solve()
```

Outline

1. Introduction
2. Basic Modeling
3. **Advanced Modeling and Search**
4. Controling Search
5. Behind the Scene

Advanced Modeling

Advanced Features

- Advanced Numberjack constructs
- Controlling search

Language constructs

Variable containers: VarArray, Matrix

- Variable Indexing (Element constraint)
- Accessors (matrix.col, matrix.flat)
- Fancy printing

Adding new constraints and decompositions

- When a constraint name is not recognized by a solver, a decomposition is used, if it exists.
- Makes it easy to add in solver specific constraints

Domain specific modules

- Dedicated constructs for Scheduling
  - Tasks
  - Resources

Sudoku

Problem Definition

- 9 x 9 grid
- Place 1 to 9 in each column, row and square.
- Pain to model; array indexing is awful!

Sudoku

Create the variables

```
grid = Matrix(N*N, N*N, 1, N*N, 'cell_')
```

The Model

```
sudoku = Model( [AllDiff(row) for row in grid.row],
                [AllDiff(col) for col in grid.col],
                [AllDiff(grid[x:x+N, y:y+N])
                 for x in range(0,N*N,N)
                 for y in range(0,N*N,N)]
               )
```

- Every row/column/sub matrix must have different digits
  - **grid.col** returns the transpose of the matrix 'grid'
  - **grid[a:b, c:d]** returns a Matrix construct that contains rows a to b-1 and columns c to d-1.

Sudoku

Adding in a partially completed grid

```
model.add ( [(x == int(v)) for (x,v)
 in zip(grid.flat, "".join(open(clues)).split() )
 if v != '*'] )
```

Isn't Python cool?!

- clues is the path to a problem file
- Entry that is not * is a constraint
- Difficult in a modeling language

```
1 * 3
4 5 *
* * 9
```

(Water retaining) Magic Square

- N x N grid
- Contains numbers 1 to N^2
- Sum of rows, columns and diagonals equal.
- Known for at least 3000 years!
- Albrecht Dürer in 1514

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

Magic Square

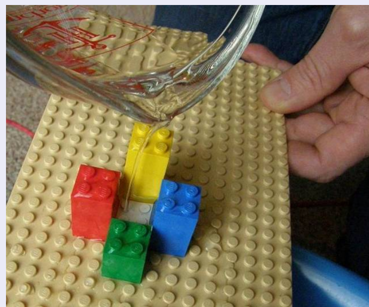
The Model

```
square = Matrix(N,N,1,N*N)
sum_val = N*(N*N+1)/2

model = Model(
    AllDiff( square ),
    [Sum(row) == sum_val for row in square.row],
    [Sum(col) == sum_val for col in square.col],
    Sum([square[a][a] for a in range(N)]) == sum_val,
    Sum([square[a][N-a-1] for a in range(N)]) == sum_val)
```

Water Retention Magic Squares

- Water retention (Craig Knecht 07)
- Magic square = 3D histogram
- Pour water on the histogram
- Maximize the amount of water



Water Retention Magic Squares

Dürer's Square

- One "lake"
- Water level = 9
- Water Depth: 3 and 2
- Water Retention = 5

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

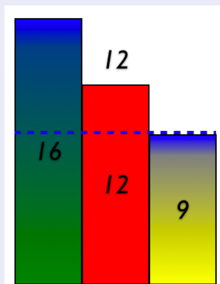
16	3	2	13
5	10	11	8
9	6+3	7+2	12
4	15	14	1

Water Retention Magic Square

How much water will it hold?

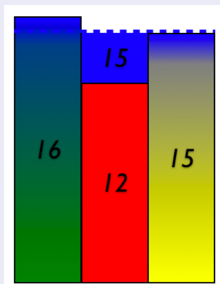
- Keep track of the total height: **sea level**

1st case: landmasses



- The "sea level" around is lower than its own height

2nd case: sea level



- The "sea level" around is greater than its own height

Water Retention Magic Square

Modeling the sea level

- Variables to represent the total level

```
water = Matrix(N,N,1,N*N)
```

Water on the edges

- The edges can't hold any water
  - Matrices allow expression of equality among variable vectors

```
water.row[0] == square.row[0]
water.row[N-1] == square.row[N-1]
water.col[0] == square.col[0]
water.col[N-1] == square.col[N-1]
```

Water Retention Magic Square

Water in the middle

- Sea level at a cell: max of its own height or the surrounding level

```
[water[a][b] ==
 Max((square[a][b],        # own height
      Min((water[a-1][b], water[a][b-1],     # sea level
           water[a+1][b], water[a][b+1])))) # on the cells
                                             # nearby
 for a in range(1,N-1)
 for b in range(1,N-1)]
```

Objective

- Goal is to maximise the water collected

```
    Maximise( Sum( water ) ) # - (N*N*(N*N-1)/2)
```

Water Retention Model

```
square = Matrix(N,N,1,N*N,'s')
water = Matrix(N,N,1,N*N,'w')
model = Model(
    Maximise( Sum( water ) ),   # objective

    # regular magic-square constraints
    AllDiff(square),
    [Sum(row) == N*(N*N+1)/2 for row in square.row],
    [Sum(col) == N*(N*N+1)/2 for col in square.col]
    Sum([square[i][i] for i in range(N)]) == N*(N*N+1)/2,
    Sum([square[i][N-i-1] for i in range(N)]) == N*(N*N+1)/2,

    # channeling with sea level
    water.row[0] == square.row[0],
    water.row[N-1] == square.row[N-1],
    water.col[0] == square.col[0],
    water.col[N-1] == square.col[N-1],
    [water[a][b] ==
     Max((square[a][b], Min((water[a-1][b], water[a][b-1],
                             water[a+1][b], water[a][b+1]))))
     for a in range(1,N-1) for b in range(1,N-1)]
    )
```

Outline

1 Introduction

2 Basic Modeling

3 Advanced Modeling and Search

4 Controling Search

5 Behind the Scene

Controling Search

Programming search & strategies

- Prototyping, search/heuristics/solvers
- Programming your own search method

Problem Solving

Simple is good

- Getting a solution quickly (Development time + Solving time)
- Prototyping, testing ideas

Parameter tuning

- Selecting
  - Variable Ordering / Branching
  - Restarts and randomization

Example (Mistral)

Problem Solving

Simple is good

- Getting a solution quickly (Development time + Solving time)
- Prototyping, testing ideas

Parameter tuning

- Selecting
  - Variable Ordering / Branching
  - Restarts and randomization

Example (Mistral)

- `solver.setHeuristic('MinDomain', 'RandomSplit')`
  Select the variable with 'min domain' first, branch by splitting the domain around a random pivot

Problem Solving

Simple is good

- Getting a solution quickly (Development time + Solving time)
- Prototyping, testing ideas

Parameter tuning

- Selecting
  - Variable Ordering / Branching
  - Restarts and randomization

Example (Mistral)

- `solver.setHeuristic('MinDomain', 'RandomSplit')`
  Select the variable with 'min domain' first, branch by splitting the domain around a random pivot
- `solver.solveAndRestart(LUBY, 300)` Set the restart policy to the Luby sequence with a base of 300

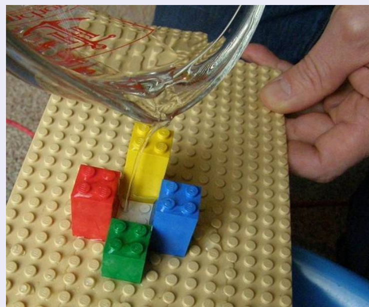
Programming Search

Search Primitives

- `solver.propagate()`: Reach a fixed point for some inference method.
- `solver.save()`: Save the current state.
- `solver.post( decision )` (`decision` is a unary constraint):
  - `x == v`, `x > v`, ...
- `solver.deduce()`: Post the complement of the decision taken at this level:
  - resp. `x != v`, `x <= v`, ... ("right" branch)
- `solver.undo()`: Restore the last saved state.

Water Retention Magic Squares

- Water retention (Craig Knecht 07)
- Magic square = 3D histogram
- Pour water on the histogram
- Maximize the amount of water



Solving the Water Retention Magic Square Problem

Challenges

- Getting good upper bounds for the objective function is hard
  - Any breach, and all the water is gone!
  - i.e., getting a good lower bound is difficult

Solving the Water Retention Magic Square Problem

Challenges

- Getting good upper bounds for the objective function is hard
  - Any breach, and all the water is gone!
  - i.e., getting a good lower bound is difficult

Solution

- There is a common sense strategy to find Magic Square that retains a large amount of water

Solving the Water Retention Magic Square Problem

Challenges

- Getting good upper bounds for the objective function is hard
  - Any breach, and all the water is gone!
  - i.e., getting a good lower bound is difficult

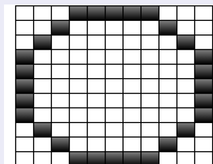
Solution

- There is a common sense strategy to find Magic Square that retains a large amount of water
  - Building a wall, i.e., high values toward the edges

Solving the Water Retention Magic Square Problem

Building a wall

- The wall should not span over a whole row/column
- We want it as high as possible
  - It makes it difficult to find a Magic Square



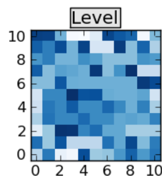
Strategy

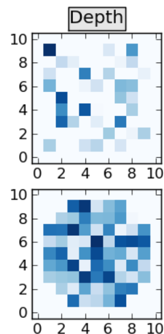
```
wall = ... # assumed to be a set of variables (cells)
for height in range(N*N-len(wall), N*N-(len(wall)+K), -1):
    solver.save()
    for brick in wall: solver.post(brick > height)
    solver.solveAndRestart()
    if solver.is_sat(): break;
    solver.undo()
```

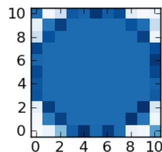
Demo!

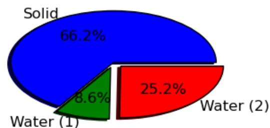
Water Retention Test

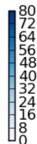
- Default search (1):
- Wall search (2):
- Water Retention:







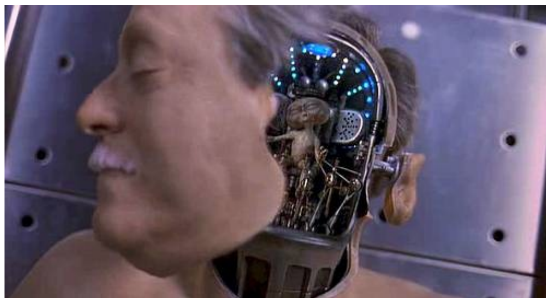




Outline

1. Introduction
2. Basic Modeling
3. Advanced Modeling and Search
4. Controling Search
5. **Behind the Scene**

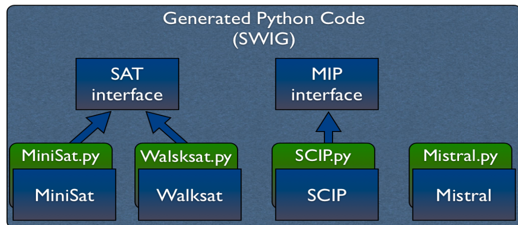
Behind the Scene



The machinery

- Wrapping a new solver
- SAT and MIP encodings

Everything can be wrapped in Python



Architecture

- C++ standardized interfaces
  - CP: Adding expression trees (direct wrapping)
  - MIP: Adding linear equations (encoder)
  - SAT: Adding clauses (encoder)
- Modules are automatically generated (SWIG)

SAT Encoding

Variables - direct/order encoding

- Two atoms for each value: $x = v$ and $x \leq v$
- A linear-sized clause: $\bigvee_{v \in D(x)} x = v$
- A linear number of 2-clauses:
  - $x \leq v \rightarrow x \leq v + 1$
  - $x = v \rightarrow x \leq v$
  - $x = v \rightarrow \neg(x \leq v - 1)$

Constraints

- Example: disequality $(x \neq y)$
  - $\forall v \in D(x) \cap D(y) \;\; \neg(x = v) \vee \neg(y = v)$
- Example: precedence $(x \leq y)$
  - $\forall v \in D(x) \setminus [-\infty, max(y)] \;\; x \leq v$
  - $\forall v \in D(y) \setminus [min(x), \infty] \;\; \neg(y \leq v)$
  - $\forall v \in D(x) \cap D(y) \;\; y \leq v \rightarrow x \leq v$

MIP Encoding

Variables

- A linear integer variable with the same bounds
- Channelled with 0/1 variables standing for $x = v$
  - *Only if required by a constraint*
  - $\sum_{v \in D(x)} (x = v) = 1$
  - $\sum_{v \in D(x)} (x = v) * v = x$

Constraints

- Example: disequality $(x \neq y)$
  - $\forall v \in D(x) \cap D(y) \quad (x = v) + (y = v) \leq 1$
- Example: Alldiff$(x_1, \ldots, x_n)$
  - $\forall v \in \bigcup_{1 \leq i \leq n} D(x_i) \quad \sum_{1 \leq i \leq n} (x_i = v) \leq 1$
  - With "domain" equations: network flow

Conclusion

Yet another platform?

- Yes, but not a new language
  - API, hence deeper control the back-end solvers
  - Python is an established programming language
  - It is easy to plug into other applications

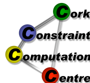
Future work

- More back-end solvers
  - CPlex
  - CP Optimizer
  - OR tools (Google CP solver)
  - ToolBar!
  - ...
- Encodings
  - Need to be more exhaustive
  - Also need research work
    - Good thesis topic!

Constraint Programming and Combinatorial Optimisation in Numberjack

Emmanuel Hebrard, Eoin O'Mahony, Barry O'Sullivan

LAAS-CNRS
Cork Constraint Computation Centre, University College Cork
This work was supported by Science Foundation Ireland (Grant No. 05/IN/I886).



December 17, 2010