Numberjack
A platform for combinatorial optimization

Barry Hurley, Emmanuel Hebrard, Eoin O’Mahony, Barry O’Sullivan

Cork Constraint Computation Centre
University College Cork, Ireland

September 16, 2013

Supported by Science Foundation Ireland Grant 10/IN.1/I3032.
Numberjack Overview

What is Numberjack?

- A platform for combinatorial optimization
  - Open source project (Github, LGPL)
- Common language for diverse paradigms (CP, SAT, MIP)
- Fast backend solvers

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 integrate into other applications
CP, SAT, MIP: choice is good!

Logically equivalent

• They are all NP-complete: there always exists a reduction

Operationally different

• Algorithms are different
  • CP: Constraint propagation + Search
  • SAT: Unit propagation + Clause Learning + Search
  • MIP: Linear relaxation + Cutting planes + Branch & Bound
• Encodings matter

Numberjack

• Language: union of SAT, MIP and CP
• Encoded depending on the choice of solver
Backend Solvers Available

### CP Solvers
- Mistral
- Toulbar

### SAT Solvers
- MiniSat
- Walksat

### MIP Solvers
- Gurobi
- CPLEX
- SCIP
- Through COIN-OR Open Solver Interface:
  - COIN-OR CBC
  - COIN-OR CLP
  - COIN-OR DyLP
  - COIN-OR SYMPHONY
  - COIN-OR Volume Algorithm
  - GNU LP Toolkit
  - Soplex
Solving the Model

Typical (simplified) Procedure

1. Specify the variables and constraints (model)
2. Pass the model to a solver
3. Set solver parameters
4. solve()
## Variables in Numberjack

### Different Variable constructors

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable()</td>
<td>Boolean variable</td>
</tr>
<tr>
<td>Variable('x')</td>
<td>Boolean variable called 'x'</td>
</tr>
<tr>
<td>Variable(N)</td>
<td>Variable in the domain of ([0 \ldots N - 1])</td>
</tr>
<tr>
<td>Variable(N, 'x')</td>
<td>Variable in the domain of ([0 \ldots N - 1]) called 'x'</td>
</tr>
<tr>
<td>Variable(l,u)</td>
<td>Variable in the domain of ([l \ldots u])</td>
</tr>
<tr>
<td>Variable(l,u, 'x')</td>
<td>Variable in the domain of ([l \ldots u]) called 'x'</td>
</tr>
<tr>
<td>Variable(list)</td>
<td>Variable with domain specified as a list</td>
</tr>
<tr>
<td>Variable(list, 'x')</td>
<td>Variable with domain as a list called 'x'</td>
</tr>
</tbody>
</table>
More Variables in Numberjack

Similarly for arrays and matrices

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VarArray(N)</td>
<td>Array of $N$ Boolean variables</td>
</tr>
<tr>
<td>VarArray(N, u)</td>
<td>Array of $N$ variables with domains $[0 \ldots u - 1]$</td>
</tr>
<tr>
<td>VarArray(N, l, u)</td>
<td>Array of $N$ variables with domains $[l \ldots u]$</td>
</tr>
<tr>
<td>Matrix(N, M)</td>
<td>$N \times M$ matrix of Boolean variables</td>
</tr>
<tr>
<td>Matrix(N, M, u)</td>
<td>$N \times M$ matrix of variables with domains $[0 \ldots u - 1]$</td>
</tr>
<tr>
<td>Matrix(N, M, l, u)</td>
<td>$N \times M$ matrix of variables with domains $[l \ldots u]$</td>
</tr>
</tbody>
</table>
Concise Models

VarArray and Matrix allow for concise modelling.

Slices
Given a $9 \times 9$ Sudoku matrix, we could get a $3 \times 3$ cell:

$\text{AllDiff( matrix}[x:x+3, y:y+3]\text{)}$

Rows and Columns

for row in matrix.row:
    m += AllDiff(row)

for col in matrix.col:
    m += AllDiff(col)
Constraints in Numberjack

Constraints can be specified in a number of ways
Simply by arithmetic operators on variables:

\[
\begin{align*}
x & > y \\
y & <= z \\
x & != z \\
x + y & == z \\
x + 4 & > z * 3 \\
z & == (x < y) \\
z & <= (x < y) \\
(x == y) & != (a == b)
\end{align*}
\]
Global Constraints

\[ v = \text{Max}(x, y, z) \]

Disjunction \([x < y, y > z, a \neq b] \)

\text{AllDiff}(x, y, z)

\[ \text{Sum}(a, b, c, d) \geq e \]

\[ \text{Sum}(a, b, c, d, [2, 1, 0.5, 3]) = e \quad \# \text{Weighted Sum} \]

\[ \text{Sum}(2a, b, 0.5c, 3d) = e \quad \# \text{Weighted Sum} \]

\text{LessLex}(\text{VarArray}(5, 5), \text{VarArray}(5, 5))

\text{LeqLex}(\text{VarArray}(5, 5), \text{VarArray}(5, 5))

\text{Gcc}(\text{VarArray}(4, 1, 5), \{1: [1, 3], 2: [1, 2]\})

\text{Element}(\text{VarArray}(10, 1, 10), \text{Variable}(10))

\[ \text{myarray}[\text{myvariable}] \quad \# \text{Element} \]

\text{Minimize}(2x + 3y)

\text{Maximize}(\text{objective})
Prototyping is useful!

One model, many solvers

- Being able to run the same model on solvers quickly and easily give fast feedback as to what seems to be the most promising approach.
- Quickly see that the warehouse location problem is solved quickly with MIP, whereas CP runs into a wall.
One Model, Many Solvers

**Constraint Solvers**

```python
mistral = model.load("Mistral")
toulbar = model.load("Toulbar2")
```

**MIP Solvers**

```python
gurobi = model.load("Gurobi")
cplex = model.load("CPLEX")
```

**SAT Solvers**

```python
minisat = model.load("MiniSat")
walksat = model.load("Walksat")
```
Configuring the Solver

# Search strategies (solver specific)
mistral.setHeuristic("DomainOverWDegree", "Lex")
mistral.setHeuristic("Impact")

# Set Limits
solver.setTimeLimit(60) # Seconds
solver.setNodeLimit(1000000)

# Solve
solver.solve()

# Solve using restarts
mistral.solveAndRestart(GEOMETRIC, 64, 1.3)
mistral.solveAndRestart(LUBY, 1000)

# Find all solutions
solver getNextSolution()

# Feasibility?
solver.is_sat()
solver.is_opt()
solver.is_unsat()
Controlling the Encoding

SAT

model.load("MiniSat", encoding=direct)
model.load("MiniSat", encoding=support)
model.load("MiniSat", encoding=order)
model.load("MiniSat", encoding=direct_order)
model.load("MiniSat", encoding=direct_support)
Controlling the Encoding

**SAT**

```python
model.load("MiniSat", encoding=direct)
model.load("MiniSat", encoding=support)
model.load("MiniSat", encoding=order)
model.load("MiniSat", encoding=direct_order)
model.load("MiniSat", encoding=direct_support)
```

**MIP**

Future work...
**Encodings Example**

Simple with three variables \( \{X, Y, Z\} \), each with domain \( \langle 1, 2, 3 \rangle \).

Constraints: \( X \neq Y \), \( X \neq Z \), and \( Y \neq Z \).

<table>
<thead>
<tr>
<th>Domain Clauses</th>
<th>Direct Encoding</th>
<th>Support Encoding</th>
<th>Order Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>((x_1 \lor x_2 \lor x_3) (\neg x_1 \lor \neg x_2) (\neg x_1 \lor \neg x_3) (\neg x_2 \lor \neg x_3))</td>
<td>((y_1 \lor y_2 \lor y_3) (\neg y_1 \lor \neg y_2) (\neg y_1 \lor \neg y_3) (\neg y_2 \lor \neg y_3))</td>
<td>((z_1 \lor z_2 \lor z_3) (\neg z_1 \lor \neg z_2) (\neg z_1 \lor \neg z_3) (\neg z_2 \lor \neg z_3))</td>
<td>((\neg x_{\leq 1} \lor x_{\leq 2}) (\neg x_{\leq 2} \lor x_{\leq 3}) (x_{\leq 3}))</td>
</tr>
<tr>
<td>(X \neq Y)</td>
<td>((\neg x_1 \lor \neg y_1) (\neg x_2 \lor \neg y_2) (\neg x_3 \lor \neg y_3))</td>
<td>((\neg y_1 \lor x_2 \lor x_3) (\neg y_2 \lor x_1 \lor x_3) (\neg y_3 \lor x_1 \lor x_2))</td>
<td>((\neg x_{\leq 1} \lor \neg y_{\leq 1}) (\neg x_{\leq 2} \lor x_{\leq 1} \lor \neg y_{\leq 2} \lor y_{\leq 1}))</td>
</tr>
<tr>
<td>(X \neq Z)</td>
<td>((\neg x_1 \lor \neg z_1) (\neg x_2 \lor \neg z_2) (\neg x_3 \lor \neg z_3))</td>
<td>((\neg z_1 \lor x_2 \lor x_3) (\neg z_2 \lor x_1 \lor x_3) (\neg z_3 \lor x_1 \lor x_2))</td>
<td>((\neg x_{\leq 1} \lor \neg z_{\leq 1}) (\neg x_{\leq 2} \lor x_{\leq 1} \lor \neg z_{\leq 2} \lor z_{\leq 1}))</td>
</tr>
<tr>
<td>(Y \neq Z)</td>
<td>((\neg y_1 \lor \neg z_1) (\neg y_2 \lor \neg z_2) (\neg y_3 \lor \neg z_3))</td>
<td>((\neg z_1 \lor y_2 \lor y_3) (\neg z_2 \lor y_1 \lor y_3) (\neg z_3 \lor y_1 \lor y_2))</td>
<td>((\neg y_{\leq 1} \lor \neg z_{\leq 1}) (\neg y_{\leq 2} \lor y_{\leq 1} \lor \neg z_{\leq 2} \lor z_{\leq 1}))</td>
</tr>
</tbody>
</table>
Example: Magic Square

Definition

“An order $N$ magic square is a $N$ by $N$ matrix containing the numbers 1 to $n^2$, with each row, column and main diagonal equal the same sum.” — CSPLib prob019

\[
\begin{array}{cccc}
16 & 3 & 2 & 13 \\
5 & 10 & 11 & 8 \\
9 & 6 & 7 & 12 \\
4 & 15 & 14 & 1 \\
\end{array}
\]
Modelling Example: Magic Square

Definition

“An order $N$ magic square is a $N \times N$ matrix containing the numbers 1 to $n^2$, with each row, column and main diagonal equal the same sum.” — CSPLib prob019

```
square = Matrix(N, N, 1, N*N)
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)
```
Solving the Magic Square model

```
python MagicSquare.py -solver Mistral
[[1, 16, 6, 11],
 [4, 13, 7, 10],
 [15, 2, 12, 5],
 [14, 3, 9, 8]]
Nodes: 17

python MagicSquare.py -solver Mistral2
[[1, 16, 10, 7],
 [13, 4, 6, 11],
 [8, 9, 15, 2],
 [12, 5, 3, 14]]
Nodes: 8

python MagicSquare.py -solver Toulbar2
[[7, 6, 11, 10],
 [14, 9, 8, 3],
 [12, 15, 2, 5],
 [1, 4, 13, 16]]
Nodes: 53

python MagicSquare.py -solver Gurobi
[[15, 14, 3, 2],
 [10, 1, 16, 7],
 [4, 11, 6, 13],
 [5, 8, 9, 12]]
Nodes: 236

python MagicSquare.py -solver CPLEX
[[14, 15, 1, 4],
 [9, 2, 16, 7],
 [8, 11, 5, 10],
 [3, 6, 12, 13]]
Nodes: 0

python MagicSquare.py -solver MiniSat
[[3, 13, 2, 16],
 [10, 8, 11, 5],
 [15, 1, 14, 4],
 [6, 12, 7, 9]]
Nodes: 24503
```
Example: Sudoku

Problem Definition

- 9 x 9 grid
- Place 1 to 9 in each column, row and square.
Modelling Example: Sudoku

\[
\text{matrix} = \text{Matrix}(N\times N, N\times N, 1, N\times N)
\]

\[
\text{sudoku} = \text{Model}\left( \left[ \text{AllDiff}(\text{row}) \text{ for } \text{row} \text{ in } \text{matrix}.\text{row} \right], \right.
\left. \left[ \text{AllDiff}(\text{col}) \text{ for } \text{col} \text{ in } \text{matrix}.\text{col} \right], \right.
\left. \left[ \text{AllDiff}(\text{matrix}[x:x+N, y:y+N].\text{flat}) \text{ for } x \text{ in } \text{range}(0,N\times N,N) \right] \right)
\]

- Every row must have different digits
- Every column must have different digits
- Every sub matrix must have different digits
Taking advantage of Python

Python is widely supported

- There are many powerful libraries available for Python
- Fast to develop and easy to use
- High level, natural language-like
Super node placement problem

- Each blue dot represents an existing node
- Place super nodes such that each existing node is covered by at least two super nodes
- Minimise number of super nodes used

Geolocation with Basemap

- Allows easy visualisation of search solutions
- Often observing solution gives intuition as to what should be done
for nn in ds:
    n=len(ds[nn])
    inv=computeInv(ds,nn)

    # all nodes need to be reached
    model += Sum([y[nn][mn][n] for j in y[nn][mn]]) == n-1

    # every node has at most one predecessor
    for i in ds[nn]:
        if i != 'nn':
            model += Sum([x[nn][mn][n] for j in inv[i]]) == 1

    # the external flow at each node is maintained
    for i in ds[nn]:
        if i != 'nn':
            in_flow=Sum([y[nn][mn][n] for j in inv[i] if j!=i])
            out_flow=Sum([y[nn][mn][n] for j in y[nn][mn] if j!=i])
            model += in_flow - out_flow == 1

    # Keeping consistency between flow and usage of the link
    for i in x[nn]:
        for j in x[nn][i]:
            model += y[nn][i][j] - n*x[nn][i][j] <= 0

    # Enforcing distance constraint
    for i in x[nn]:
        for j in x[nn][i]:
            model += (x[nn][i][j])==0 | (l[nn][j] >= l[nn][i] + ds[nn][i][j])

    # Restricting the capacity of the link
    if capa:
        for i in y[nn]:
            for j in y[nn][i]:
                model += y[nn][i][j] <= capa

    # Enforcing disjointness
    for j in ess:
        if j in {nn1,nn2}==ess[j]:
            inv.nn1=computeInv(ds,nn1)
            inv.nn2=computeInv(ds,nn2)
            for i in (set(inv.nn1[j]) & set(inv.nn2[j])):
                model += <expression> <expression> <expression> <expression> <expression> <expression> <expression> <expression> <expression> <expression>
Recent Improvements

What’s new?

- Added Toulbar2
- Added Gurobi
- Added ILOG CPLEX
- Integration with COIN-OR Open Solver Interface solvers
- Minizinc/Flatzinc input
- Configurable encodings
- Many bug-fixes
What is Numberjack?

• A platform for combinatorial optimization
• Open source project (Github, LGPL)
• Common language for diverse paradigms (CP, SAT, MIP)
• Fast backend solvers
• Easy to integrate into other applications

http://numberjack.ucc.ie/
numberjack.support@gmail.com