Outline

Introduction to Combinatorial Optimization

Modelling

Solving Technologies

Modelling and Solving with Numberjack

Summary
Section 1

Introduction to Combinatorial Optimization
About this Tutorial

Motivation
Combinatorial optimization provides powerful support for decision-making; many interesting real-world problems are combinatorial and can be approached using similar techniques.
About this Tutorial

Motivation
Combinatorial optimization provides powerful support for decision-making; many interesting real-world problems are combinatorial and can be approached using similar techniques.

The Promise
This tutorial will teach attendees how to develop interesting models of combinatorial problems and solve them using constraint programming, satisfiability, and mixed integer programming techniques.
What is Combinatorial Optimisation?
Optimization problems divide neatly into two categories, those involving:

- continuous variables
- discrete variables -- these are *combinatorial*.

Usually we’re concerned with finding a least-cost solution to a set of constraints.

Our focus in this tutorial

- Constraint Programming
- Satisfiability
- Mixed Integer Programming
What is a Combinatorial Problem?

Variables, Domains, and Constraints
Given a set of variables, each taking a value from a domain of possible values, find an assignment to all variables that satisfy the constraints.

<table>
<thead>
<tr>
<th>Variables</th>
<th>Domains</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adjacent counties must be coloured differently.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What is a Combinatorial Problem?

Variables, Domains, and Constraints
Given a set of variables, each taking a value from a domain of possible values, find an assignment to all variables that satisfy the constraints.

<table>
<thead>
<tr>
<th>Variables</th>
<th>Domains</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adjacent counties must be coloured differently.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Well known example: Sudoku

Variables and Domains
- Each cell must take a value from 1 to 9.

Constraints
- All cells in a row must be different.
- All cells in a column must be different.
- Each $3 \times 3$ block must be different.
Applications: Optical Network Design

- Configuration of the backhaul network
- Every exchange must be connected to a metro node
- Every exchange must be reachable along two paths
- Paths are bounded in length
- Links are of limited capacity
- The amount of cable should be minimised
Applications: Rosetta/Philae Mission

Scheduling Scientific Experiments on the Rosetta/Philae Mission

_Gilles Simonin, Christian Artigues, Emmanuel Hebrard, Pierre Lopez_

Image: ESA
price = [215, 275, 335, 355, 420, 580]
apitzers = ['Mixed Fruit', 'French Fries', 'Side Salad', 'Hot Wings', 'Mozzarella Sticks', 'Sample Plate']

quantities = VarArray(len(apitzers), 0, max_qty)
model = Model( Sum(quantities, price) == 1505 )
Types of Applications

Satisfaction

- Find a single solution
- Example: conference schedule

Optimization

- Find the best/high quality solution
- Minimize/maximize: cost, time, profit

Many many others

Scheduling; planning; configuration; nurse rostering; timetabling; vehicle routing; electricity, water, and oil networks; sourcing optimization; bioinformatics; . . .
Section 2

Modelling
Modelling Languages

Features
Declarative specification of the problem, separating (in so far as possible) the formulation and the search strategy.

A Constraint Model of the Sudoku Puzzle

matrix = Matrix(N∗N, N∗N, 1, N∗N)

sudoku = Model(
    [AllDiff(row) for row in matrix.row],
    [AllDiff(col) for col in matrix.col],
    [AllDiff(matrix[x:x+N, y:y+N]) for x in range(0, N∗N, N) for y in range(0, N∗N, N)]
)
Modelling Languages

Features
Declarative specification of the problem, separating (in so far as possible) the formulation and the search strategy.

A Constraint Model of the Sudoku Puzzle

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

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

Problem

Human

Model

Solver

Solution
Modelling

• This is *one* model, not *the* model of the problem.
• Many possible alternatives
• Not always clear which is the *best* model
• Often: not clear what is the *problem*
More Realistic

- Problem
- Human
- Model
- Solver
- Hangs
- Solution
- Wrong Solution
Modelling Frameworks

Benefits

- High-level, solver independent model
- Use of multiple back-end solvers
- Automatic translation for each solver

Some Modelling Frameworks

- Numberjack (Insight, Ireland)
- Savile Row (St Andrews, UK)
- Minizinc (NICTA, Australia)
Framework Process

Problem

Human

Model

Compile/Reformulate

CP

Solution

MIP

Solution

SAT

Solution

Other

Solution
Section 3

Solving Technologies
Solving a Combinatorial Problem

Many possibilities

- Constraint Programming (CP)
- Satisfiability (SAT)
- Mixed Integer Programming (MIP)
- Local Search -- heuristic guess with heuristic repair
- Large Neighbourhood Search -- systematic and local search
Solving a Combinatorial Problem

Many possibilities

- **Constraint Programming (CP)**
- **Satisfiability (SAT)**
- **Mixed Integer Programming (MIP)**
- Local Search -- heuristic guess with heuristic repair
- Large Neighbourhood Search -- systematic and local search
Constraint Programming (CP)

In a nutshell

- Domains typically represented as integers (others: real valued, set, graph)
- Typically solved using backtracking-\textit{search}
- Polynomial-time \textit{inference} reduces the size of the search space

Inference/Propagation

The All-Different constraint specifies that variables must be assigned distinct values.

\begin{align*}
x_1 &\in \{1, 2, 5\} \\
x_2 &\in \{1, 2, 5\} \\
x_3 &\in \{1, 2, 5\} \\
x_4 &\in \{2, 3, 4\} \\
x_5 &\in \{1, 2, 3, 4, 5\}
\end{align*}
Constraint Programming (CP)

In a nutshell

- Domains typically represented as integers (others: real valued, set, graph)
- Typically solved using backtracking-\textit{search}
- Polynomial-time \textit{inference} reduces the size of the search space

Inference/Propagation

The All-Different constraint specifies that variables must be assigned distinct values.

\begin{align*}
\mathbf{x}_1 & \in \{1, 2, 5\} & \quad \mathbf{x}_1 & \in \{1, 2, 5\} \\
\mathbf{x}_2 & \in \{1, 2, 5\} & \quad \mathbf{x}_2 & \in \{1, 2, 5\} \\
\mathbf{x}_3 & \in \{1, 2, 5\} & \quad \mathbf{x}_3 & \in \{1, 2, 5\} \\
\mathbf{x}_4 & \in \{2, 3, 4\} & \quad \mathbf{x}_4 & \in \{3, 4\} \\
\mathbf{x}_5 & \in \{1, 2, 3, 4, 5\} & \quad \mathbf{x}_5 & \in \{3, 4\}
\end{align*}
Sudoku: simple example of CP propagation

<table>
<thead>
<tr>
<th>Initial domains with preset clues.</th>
<th>After propagating the preassigned cells.</th>
<th>Complete solution after search.</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Initial Sudoku Grid" /></td>
<td><img src="image" alt="Propagated Sudoku Grid" /></td>
<td><img src="image" alt="Complete Solution Grid" /></td>
</tr>
</tbody>
</table>

**Core Idea**

Remove inconsistent values from domains which cannot be part of a solution.
Satisfiability Problem (SAT)

- Set of Boolean variables $x_1, x_2, \ldots, x_n$
- A literal is a variable $x_1$ or its negation $\neg x_1$
- A clause is a disjunction of literals $(x_2 \lor \neg x_3)$
- Formula e.g. $(x_1 \lor x_3 \lor \neg x_4) \land (x_4) \land (x_2 \lor \neg x_3)$

- A clause is satisfied when at least one of its literals is \textit{true}.
- A formula is satisfied when all clauses are satisfied.
- \textbf{Goal:} Find an assignment to the variables that satisfies the formula
SAT Solving

- Backtracking search
  - 1\textsuperscript{st} decision: \(a\)
    - \((\neg a \lor \neg b)\)
  - 2\textsuperscript{nd} decision: \(c\)
    - \((b \lor \neg c \lor d)\)
    - \((\neg d \lor e)\)
  - 3\textsuperscript{rd} decision: \(f\)
    - \((b \lor \neg f \lor g)\)
    - \((\neg a \lor \neg f \lor h)\)

- Conflict-driven clause learning
  - \((\neg d \lor \neg i)\)
Mixed Integer Programming

Standard MIP Model has the following form

\[ \min cx + dy \]  
\[ \text{s.t.} \quad Ax + By \geq 0 \]  
\[ x, y \geq 0 \]  
\[ y \text{ integer} \]

Informally

The objective function is linear, e.g. a weighted sum, expression over the variables. Each constraint is linear. Some variables take integer values, other can take real values.
MIP Solving

![Graph showing a grid with points at integer coordinates from (1,1) to (6,6).]
MIP Solving
MIP Solving
MIP Solving

Fractional solution
MIP Solving

Fractional solution

Cut

Insight Centre for Data Analytics PyCon Ireland 2014 Slide 30
CP, SAT, MIP: choice is good!

Operationally different

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

- Often not clear which is best for a problem
- Encodings matter

Numberjack

- Write a single model in Python
- Numberjack constructs: union of SAT, MIP, and CP
- Encoded depending on the choice of solver
- Rapidly evaluate different solvers and paradigms
Summary of Solving Technologies

Core goal

- The user states the problem and a general purpose solver is used to find a solution
- Generic tools to solve a broad range of applications

Solving Approaches

- Multiple complimentary approaches: CP, SAT, MIP
- Often not clear which is best for a particular problem
- Choice is good, modelling frameworks allow to try many possibilities
Section 4

Modelling and Solving with Numberjack
Numberjack Overview

What is Numberjack?

- A Python platform for combinatorial optimization
- Open source project (Github, LGPL)
- Common platform for diverse paradigms (CP, SAT, MIP)
- Fast backend solvers
# Backend Solvers Available

## CP Solvers
- Mistral v1 & v2
- Toulbar2

## SAT Solvers
- MiniSat
- Walksat
- Lingeling
- CryptoMiniSat
- Glucose
- ... *any CNF solver*

## MIP Solvers
- IBM ILOG CPLEX
- Gurobi
- 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
Installation

Python Package Index (PyPI)

pip install Numberjack
Subsection 1

Modelling Constructs
### Different Variable constructors

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

Constraints can be specified by arithmetic operators of variables

\[
\begin{align*}
    x &> y \\
    y &\leq z \\
    x &\neq z \\
    x + y &\equiv z \\
    x + 4 &> z \times 3 \\
    z &\equiv (x < y) \\
    z &\leq (x < y) \\
    (x \equiv y) &\neq (a \equiv b)
\end{align*}
\]
Hello World Example

```python
from Numberjack import *

x, y, z = VarArray(3, 5)  # 3 variables with domains 0..4
model = Model(
    x != y, x != z, y != z,
    x == y + z,
    z > y,
)

solver = model.load("Mistral")  # Load the model into a solver
solver.solve()  # Tell the solver to solve

print x.get_value(), y.get_value(), z.get_value()  
```

Output:
from Numberjack import *

x, y, z = VarArray(3, 5)  # 3 variables with domains 0..4
model = Model(
    x != y, x != z, y != z,
    x == y + z,
    z > y,
)

solver = model.load("Mistral")  # Load the model into a solver
solver.solve()  # Tell the solver to solve

print x.get_value(), y.get_value(), z.get_value()

Output: 3 1 2
Global Constraints

Many benefits

- Constraints over multiple variables
- More concise modelling
- More expressive
- Often specialized algorithms to achieve higher levels of inference.

All-Different
Each variable must take a distinct value.

# We should write:
AllDiff([w, x, y, z])

# instead of:
w != x, w != y, w != z, x != y, x != z, y != z
Concise Models

VarArray and Matrix allow for concise modelling.

Rows and Columns

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

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

Slices

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

```python
AllDiff( matrix[x:x+3, y:y+3] )
```
Global Constraints: Sum

\[ \text{Sum}(\text{row}) = 1 \]

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

**Weighted Sum**

\[ 2a + b + 0.5c + 3d = e \]

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

\[ \text{Sum}([a, b, c, d], [2, 1, 0.5, 3]) = e \]
Global Constraints: Element

**Element**
Allows indexing into a variable array (at solve time) by the value of another variable. Useful modelling construct.

**Numberjack**

```python
x == Element(VarArray(10, 1, 10), Variable(10))

y == myvararray[myvariable]  # Element
```
Custom Constraints

Easy to define your own custom constraints for modelling purposes and add custom decompositions for solvers which don’t support it.

Example

class MyAllDiff(Expression):

    def decompose(self):
        return [var1 != var2 for var1, var2 in pair_of(self.children)]

class NoOverlap(Predicate):

    def decompose(self):
        task_i, task_j = self.children
        return [((task_i + task_i.duration) <= task_j) |
                ((task_j + task_j.duration) <= task_i)]
Optimizing: Minimize and Maximize

Objectives

- Finding the best/good solutions.
- e.g. minimise cost, wastage, loss, time; maximize profit.

Numberjack

Minimize($2x + 3y$)

Maximize(objectivevariable)
Subsection 2

Solving
Framework Process

Problem

Human

Model

Compile/Reformulate

CP

MIP

SAT

Other

Solution

Solution

Solution

Solution
One Model, Many Solvers

**Constraint Solvers**

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

**MIP Solvers**

gurobi = model.load("Gurobi")
cplex = model.load("CPLEX")
scip = model.load("SCIP")

**SAT Solvers**

minisat = model.load("MiniSat")
walsat = model.load("Walksat")
# Search strategies (solver specific)
mistral.setHeuristic("DomainOverWDegree", "Lex")
mistral.setHeuristic("Impact")

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

# Solve
solver.solve()

# Find all solutions
solver.getNextSolution()

# Feasibility?
solver.is_sat()
solver.is_opt()
solver.is_unsat()
from Numberjack import *
N = 3
matrix = Matrix(N*N, N*N, 1, N*N)
model = Model(
    [AllDiff(row) for row in matrix.row],
    [AllDiff(col) for col in matrix.col],
    [AllDiff(matrix[x:x+N, y:y+N])
      for x in range(0, N*N, N) for y in range(0, N*N, N)]
)
solver = model.load("Mistral")  # Load the model into a solver
solver.solve()  # Tell the solver to solve

if solver.is_sat():  # Found a solution
    print matrix
elif solver.is_unsat():  # Solver proved that no solution exists
    print "Unsatisfiable"
Subsection 3

Modelling Examples
Magic Square

- $N \times N$ grid
- Contains numbers 1 to $N^2$
- Sum of rows, columns, and diagonals equal the same value.

$$\frac{N \times (N^2 + 1)}{2}$$

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

The Model

# N x N matrix of variables from 1..N^2
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)
N-Queens

Problem Definition
Place $N$ queens on an $N \times N$ chessboard so that no queen attacks another. A queen attacks all cells in horizontal, vertical and diagonal direction.

One solution for $N = 8$
Basic N-Queens Model

Cell based model

- A 0/1 variable for each cell to say if it is occupied or not
- Constraints on the rows, columns, and diagonals to enforce no attach
- $N^2$ Boolean variables, $6N - 4$ constraints

Column (row) based model

- Each column (row) must contain exactly one queen
- A 1..$N$ variable for each column representing the position of the queen in the column
- $N$ variables, 3 global constraints (or $N^2/2$ binary constraints)
N-Queens Model

# One variable for each queen (one per column) representing # the row in which that queen is placed.
queens = VarArray(N, N)

model = Model(
    # Queens must be placed in different rows
    AllDiff(queens),

    # Queens are not on the same diagonals
    AllDiff([queens[i] + i for i in range(N)]),
    AllDiff([queens[i] - i for i in range(N)])
)

Insight Centre for Data Analytics PyCon Ireland 2014 Slide 58
Golomb Ruler

Problem definition

- Place $N$ marks on a ruler
- Distance between each pair of marks is different
- Goal is to minimise the placement of the last mark
Golomb Ruler

Create the Variables

marks = VarArray(nbMarks, rulerSize)

• Each variable represents the position of a mark

Now the model

• Modelling choices are important
• We will illustrate this with a series of models
First Model

Naive Model

model = Model(
    Minimise( Max(marks) ),
    AllDiff(marks),
    AllDiff([first - second for first, second in pair_of(marks)])
)

- Each mark must be at a different position
- Each pair of distances must be different
- Minimise the position of the last mark
First Model Analysis

This is not a very good model!

- The marks must all be different
- But they can also be totally ordered
- Instead of marks[0] being any mark on the ruler we can constrain it to be the first mark
- Then we just have to minimise the position of the last mark in the marks array
First Model Analysis

This is not a very good model!

- The marks must all be different
- But they can also be totally ordered
- Instead of marks[0] being any mark on the ruler we can constrain it to be the first mark
- Then we just have to minimise the position of the last mark in the marks array

```python
model = Model(
    Minimise( marks[−1] ),
    [marks[i] < marks[i+1] for i in range(nbMarks−1)],
    AllDiff([first − second for first, second in pair_of(marks)])
)
```
Why does this work?

These extra constraints improve the speed

- These constraints reduce both the time taken and nodes searched

Adding in constraints can help

- Stronger constraints reduce the search space
- Still maintain (a set of) solutions
More modeling advances

What can we reason about the first mark?

- It must always be at zero
- Proof: Any solution with the first mark at position $n$ can be be shifted $n$ positions to the left while maintaining all the constraints and reducing the objective function
- Obvious to us but not to the solver

```python
model = Model(
    marks[0] == 0,
    Minimise( marks[-1] ),
    [marks[i] < marks[i+1] for i in range(nbMarks-1)],
    AllDiff([first - second for first, second in pair_of(marks)])
)
```
Where are we searching?

What are the decision variables?

- By default the solvers will search on all variables in a problem
- In the Golomb ruler most of these will be auxiliary variables representing the distance between the marks
- These are functionally dependent on the positions of the marks
- So why bother searching on these!?
  
  \[
  s = \text{model.load("Mistral", marks)}
  \]
- Tell the solver what variables to search on
Modelling and Solving Summary

Modelling

- Many possible modelling alternatives
- Not always clear which is the best model
- Often: not clear what is the problem

Solving

- Often not clear which solving technology to use
- Try each alternative, easy in a modelling framework
- Search strategy can have a huge impact on performance
Map Colouring Demo

Demo
Map Colouring Demo

I learned it last night! Everything is so simple!
Hello World is just print "Hello, world!"

I dunno...
Dynamic typing?
Whitespace?
Come join us!
Programming is fun again!
It's a whole new world up here!
But how are you flying?

I just typed
import antigravity
That's it?
... I also sampled
everything in the
medicine cabinet
for comparison.
But I think this is the Python.
Summary

Numberjack

- A Python platform for combinatorial optimization
- Common platform for diverse paradigms (CP, SAT, MIP)
- Practical, easy to use, and intuitive
- Fast backend solvers
- Open source (Github, LGPL)

http://numberjack.ucc.ie
Acknowledgements

This work is supported by Science Foundation Ireland (SFI) Grant 10/IN.1/I3032.
The Insight Centre for Data Analytics is supported by SFI Grant SFI/12/RC/2289.
Links

Learn more

- Much more which we cannot cover here
- ECLiPSe ELearning Course
  http://4c.ucc.ie/~hsimonis/ELearning/index.htm
- Discrete Optimization on Coursera

Research

- Insight Centre for Data Analytics
  https://www.insight-centre.org
- UCC Research Vacancies
  http://www.ucc.ie/en/hr/vacancies/research/

Industrial Collaborations

- Chrys Ngwa, Business Development Manager, Insight
  chrys.ngwa@insight-centre.org
Exercise: Send More Money

A Crypt-Arithmetic Puzzle

- Each character stands for a digit from 0 to 9
- Numbers are built from digits in the usual, positional notation
- Repeated occurrence of the same character denote the same digit
- Different characters denote different digits

\[
\begin{align*}
\text{SEND} & + \text{MORE} \\
\hline
\text{MONEY} &
\end{align*}
\]