# COMP90046 Constraint Programming

Peter Stuckey

# Overview

- Staff
- Critical Information
- Subject Overview
- Resources
- Workload
- Getting Help
- Assessment
- Checklist

# Staff

- Lecturer: Peter J. Stuckey
  - room 6.19 Doug McDonnell
  - pstuckey@unimelb.edu.au
- Tutor: Nick Downing
  - downing.nick@gmail.com
- Guest Lecturer: Mark Wallace

# Critical Information #1

- Lecture times:
  - Mondays 12:00 - 13:00
  - Alan Gilbert 109 (Theatre 2)

- Workshop times:
  - Tuesdays 11:00 - 12:00 Alice Hoy 101
  - Fridays 11:00 - 12:00 Alice Hoy 3.33
  - workshops commence in week 2

# Critical Information #2

- Flipped classroom Coursera Course
  - Enrol in Coursera using unimelb account
  - We will enrol you in
    - [Basic Modeling Discrete Optimization]
    - [Advanced Modeling Discrete Optimization]
- Each week
  - watch the lecture videos before Monday
  - attempt the workshop questions
- Mondays lecture will involve
  - questions to determine your understanding
  - fill in / revisions / your questions answered
  - group activities to support the material

# Subject Overview

- Constraint Programming will focus on:
  - modelling combinatorial optimisation problems, and
  - technologies to solve these models

- Skills to be learned
  - Model a complex constraint problem using a high level modeling language
  - Define and explore different search strategies for solving a problem
  - Explain how modelling interacts with the solving algorithms, and formulate models to take advantage of this
  - Use state of the art optimisation tools

# Programming Environment

- We will use the modelling language
  - MiniZinc
- We will make use of the MiniZinc 2.1.x
  - www.minizinc.org/
- We will also use the
  - MiniZincIDE integrated development environment
- You can MiniZinc with multiple solvers
  - Gecode www.gecode.org
  - Chuffed https://github.com/chuffed/chuffed

# Modelling is “hands on” …

- You learn to model by
  - modelling!

- You need to write many models during semester to develop your knowledge of modelling techniques

- Practice, practice, practice

# Texts

- There is no prescribed textbook
- There is a detailed tutorial for MiniZinc
  - http://www.minizinc.org/downloads/doc-latest/minizinc-tute.pdf

- Texts on Constraint Programming
  - Programming with Constraints: an Introduction, Marriott and Stuckey, MIT Press. 1998.
  - Principles of Constraint Programming. Krzysztof Apt. Cambridge. 2003.
  - The OPL Optimization Programming Language. Pascal Van Hentenryck, MIT Press. 1999.

# Seeking Assistance

- There are a range of mechanisms to use when you need help.
  - Check the LMS site for general announcements.
  - Post your query to the LMS discussion forum. Read other posts and responses while you wait for a response to your query.
  - Ask a question during/after a lecture
  - Make an appointment to see the lecturer (send email to fix a time), or come during office hours.

# Assessment

- You final mark is a combination of two components
  - Exam: 70 marks
    - you must obtain at least 35/70 to pass the subject
  - Projects: 30 marks
    - eight projects over the course
      - more than 30 marks available
      - you don't need to do all projects
      - no project hurdle

# Assignments

- Assignments will be marked online
- Assignments are managed through the Coursera interface using your University of Melbourne login
- You will use this to submit assignments
- 8 assignments throughout the course

# Academic Honesty

- All assessed work in this subject is individual.

- It is easy for us to run sophisticated similarity checking software over all submissions.

- The University's Academic Honesty policy will be applied if duplicate work is detected.

# Pre Course Survey

- Throughout the course we will use
  - Quickpoll
- to determine your understanding of material

- We will use it now for a
  - Pre Course Survey

- http://qp.unimelb.edu.au/pstuckey

# Pre Course Survey 1A

- Prior knowledge of maths
  - A: high school only
  - B: some tertiary
  - C: significant tertiary
  - D: undergrad maths degree
  - E: postgrad maths degree

# Pre Course Survey 1B

- Prior knowledge of operations research (LP, MIP, duality, convex optimization, …)
  - A: none
  - B: beginner
  - C: intermediate
  - D: skilled
  - E: expert

# Pre Course Survey 1C

- Prior knowledge of computer science
  - A: none
  - B: since I started an MIT only
  - C: some tertiary
  - D: undergrad CS degree
  - E: postgrad CS degree

# Pre Course Survey 1D

- Prior knowledge of computer programming
  - A: none
  - B: beginner
  - C: intermediate
  - D: skilled
  - E: expert

# Pre Course Survey 1E

- Prior knowledge of declarative programming (Prolog, ML, Haskell, …)
  - A: none
  - B: beginner
  - C: intermediate
  - D: skilled
  - E: expert

# Pre Course Survey 1F

- Prior knowledge of mathematical modelling
  - A: none
  - B: beginner
  - C: intermediate
  - D: skilled
  - E: expert

# Pre Course Survey 1G

- Have you complete Declarative Programming COMP30020/COMP90048
  - A: COMP30020
  - B: COMP90048
  - C: neither
  - D: both?
  - E:

# Pre Course Survey 1H

- Have you received an invite to the Coursera Private Cohort Session
  - A: yes
  - B: no
  - C:
  - D:
  - E: expired

# Pre Course Survey 1I

- Have you downloaded and installed MiniZinc IDE
  - A: yes, checked it works
  - B: yes
  - C: downloaded it
  - D: no
  - E: what are you talking about?

# What is Discrete Optimization

- Optimize
  - Make the best or most effective use of (a situation or resource):

- Optimization: can mean a lot of things!
  - in compilers: producing code that goes faster
  - in engineering: changing a process to be more efficient
  - in mathematics:
    - finding the best possible value of some function

# Mathematical Expressions

- In this course we will assume some basic maths familiarity

- What does $x = x + 1$ mean?

- What does $x \leq y \land y \leq x$ mean?

- What does $x = 1 \lor x = -1$ mean?

- And $(x = y \lor x = -y) \land x \geq y \land x \geq -y$ ?

# Maths Notation

- variables: e.g. $x$, $y$, $z$, …
  - placeholder for a single value!
- terms: e.g. $x + 2 * y / (z - 3)$
  - built using operators +, *, -, /, etc
- sets: {1,3,4,5}, { 0.99, 23.0, 105.7 }
- ranges: e.g.  1 ..  5,   -1.5 .. 0.71
  - sets of integers or reals
    - 1.. 5  =  {1,2,3,4,5}
    - -1.5 .. 0.71 = $\{ r \in \mathbb{R} \mid -1.5 \leq r \leq 0.71 \}$
- set terms: {}, $S \cup R$, $S \cap R$
  - using empty set, union, intersection

# Maths Notation

- constraints: e.g. $x = y$, $x \leq y$, $x \in S$, $S \subseteq R$
  - built using relations $=$, $\leq$, $\subseteq$, etc.
- formulae: e.g. $(x = y \vee x = -y) \wedge x \geq y \wedge x \geq -y$
  - built using:
    - and (conjunction): $\wedge$
    - or (disjunction) $\vee$
    - iff (if and only if) $\leftrightarrow$
    - (implication) $\rightarrow$

# What is Discrete Optimization

- Mathematical Optimization (Minimization)
  - given a function $f$ from $D$ to $\mathbb{R}$
  - set of possible values $X \subseteq D$
  - find the value $x \in X$ which minimises $f(x)$
  - e.g. minimize $\cos(x)$ where $x \in [-\pi/3 .. \pi/4]$
  - e.g. minimize $x + y$ where $(x,y) \in \{ (a,b) \mid a \in 0.5 .. 25 , b \in 0.0 .. 4.3, a + 2b \geq 18 \}$
- Discrete Optimization
  - variables in $D$ take a discrete set of values
  - e.g. minimize $x + y$ where $(x,y) \in \{ (a,b) \mid a \in 1.. 25 , b \in 0 .. 4, a + 2b \geq 18 \}$

# Who cares about Discrete Optimization

- Most of the most common problems we want to optimise are discrete
  - scheduling the trains
    - we schedule trains to leave on the minute, not to the millisecond
  - rostering nurses
    - we roster nurses onto discrete shifts
  - routing trucks
    - we decide which of a finite set of customers each truck will visit
  - managing share portfolios
    - we buy shares in unit quantities
  - etc, …

# What is Modelling

- Capturing the problem we are trying to solve
  - mathematically
  - precisely (or at least to some level of detail)
  - usually so that some software can solve it

- Usually a tiny part of an IT project
  - but the crucial part
  - without it nothing else works!

# Project Lifecycle

- Identification of IT/IS opportunity
- Identification of DO opportunity
- Exploration of DO opportunity
  - rapid prototype models
- Full requirements study
  - problem definition document (models in English)
- Implementation
  - problem solution document, DO models
- Delivery/Integration
- Maintenance

# The Zinc Mantra

- Modelling and solving Discrete Optimization problems is hard
- There are many ways to solve these problems
- We should
  - model them once
    - rapidly
  - solve them
    - with many different technologies
    - in many different ways
    - rapidly

# MiniZinc

- A high level solver independent modelling language
  - subset of Zinc
  - maps to FlatZinc: a solver input language
- Supported by
  - most Constraint Programming solvers
  - some MIP solvers
  - SAT and SAT modulo theory based solvers

# Syllabus

- Syllabus
  - What is discrete optimisation
  - Basic modelling
  - Modelling sets and functions
  - Functions and predicates
  - Debugging models
  - Scheduling and packing
  - Flattening (How MiniZinc works)
  - CP solving (and programming search)
  - MIP solving
- Important
  - We cover more than the two Coursera courses

# Checklist

- Things to be done
  - Check you can access the LMS page
  - Read the course handout
  - Download and MiniZinc and the MiniZincIDE
    - www.minizinc.org
  - Enrol in Coursera (www.coursera.org) using your unimelb account
  - Enrol in the private session (you should have been emailed an invitation)
    - if no invite, please email me with your name, student id, and unimelb email

# Overview

- Staff
- Critical Information
- Subject Overview
- Resources
- Workload
- Getting Help
- Assessment
- Checklist