Constraint Systems

Lab 2 - A Few Python Tricks

Flattening a Dictionary

The search configuration API of or-tools expects a list.

```
decision_builder = slv.Phase(all_vars, ...)
```

But sometimes it's easier to store variables in a dictionary. E.g.

```
x = {(i,j): slv.IntVar(1, n) for i in range(n)
                             for j in range(n)}
```

You can "flatten" a dictionary using the **`values()`** method:

```
decision_builder = slv.Phase(x.values(), ...)
```

- **`values()`** returns the list of values in the dictionary

Generator Objects

Many functions that operate over collections (e.g. list comprehensions):

```
sum([2*i for i in range(n)])
min([2*i for i in range(n)])
...
```

...can work also with generator objects:

```
sum(2*i for i in range(n)) # no brackets, or just ()
```

- Comprehensions build a new data structure
- Generator objects produce elements "on the fly"

```
print [i for i in range(3)] # [0, 1, 2]
print (i for i in range(3)) # <generator object...
```

"Join" Method

A useful method for printing collections:

```
<string>.join(<string collection or generator object>)
```

- Merges all strings in the collection/generator object
- The first string is used as a separator

```
', '.join(['two', 'strings']) # "two, strings"
'/'.join('%d' % i for i in range(4)) # 0/1/2/3
```

Problems vs Instances

A useful (and important) distinction:

- **Problem:** a parametric description of a CSP
- **Instance:** a problem + values for all the parameters

In the last lecture:

- Each script was solving a specific instance

In this lecture:

- Each script will work for a certain problem
- The script will read instance data from external files

We will store instance data using JSON

JSON in Python

JSON is a format for data serialization

- Based on Javascript
- Primitive types: numbers, strings ("" quotes), booleans (true/false)
- Two types of collection:
  - Lists: `[item1, item2, ...]`
  - Dictionaries: `{"key1":v1, "key2":v2, ...}`

The root element in a JSON document is always a dictionary

Why JSON for our data?

- Easy to read and interpret
- Translates directly to Python data structures

JSON in Python

Example:

```
{
 "lectures": [
  { "num": 25, "long": false },
  { "num": 22, "long": false },
  { "num": 15, "long": true }
 ],
 "rooms": [
  { "cap": 66 },
  { "cap": 58 }
 ]
}
```

JSON in Python

How to read JSON data in python:

```
import json # import the "json" module

with open(fname) as f: # "f" = file object
    data = json.load(f) # read and parse JSON data
```

- `open(fname)` returns a file object
- `with` takes care of closing the file once the block is over

Syntax:

```
with <func. call> as <name of returned obj>:
    <block>
```

Constraint Systems

Lab 2 - More About the or-tools API

Search Monitors

In or-tools, we can attach monitors to the search process:

```
slv.NewSearch(decision_builder, [m1, m2, ...])
```

A search monitor is an object that "does something" during search

- Print information
- Store a solution
- Stop search under certain conditions
- ...

Search monitors are very useful

E.g. search limits are essential, since we often solve NP-hard problems

Search Monitors

In or-tools, we can attach monitors to the search process:

```
slv.NewSearch(decision_builder, [m1, m2, ...])
```

A search monitor is an object that "does something" during search

Two practical examples:

- Print some information every N branches:

```
m1 = slv.SearchLog(num_branches)
```

- Stop search after N milliseconds:

```
m2 = slv.TimeLimit(time_limit)
```

Solver Performance

Two important metrics for the performance of a CP approach:

Solution time:

- The most important metric in practice
- Measures the effectiveness of search and propagation
- Measures the cost of search and propagation

In or-tools:

```
slv.WallTime()
```

Solver Performance

Two important metrics for the performance of a CP approach:

Number of branches:

- Measures only the effectiveness of search and propagation
- System independent
- Alternative: number of failures

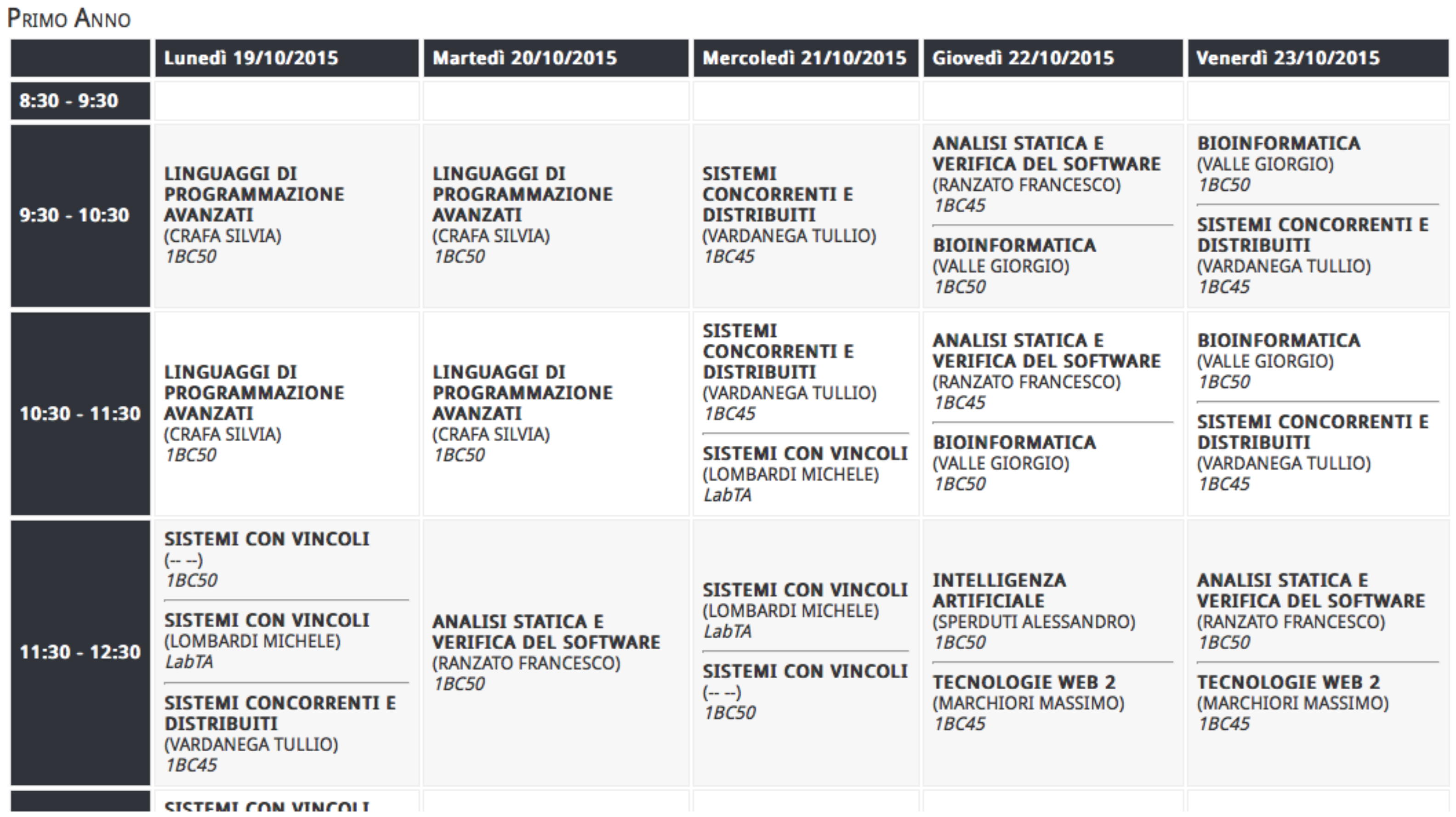
```
slv.Branches()
slv.Failures()
```

Constraint Systems

Lab 2 - A Timetabling Problem

Problem Description

A number of lectures should be held in a number of rooms.



PRIMO ANNO

	Lunedì 19/10/2015	Martedì 20/10/2015	Mercoledì 21/10/2015	Giovedì 22/10/2015	Venerdì 23/10/2015
8:30 - 9:30					
9:30 - 10:30	**LINGUAGGI DI PROGRAMMAZIONE AVANZATI** (CRAFA SILVIA) *1BC50*	**LINGUAGGI DI PROGRAMMAZIONE AVANZATI** (CRAFA SILVIA) *1BC50*	**SISTEMI CONCORRENTI E DISTRIBUITI** (VARDANEGA TULLIO) *1BC45*	**ANALISI STATICA E VERIFICA DEL SOFTWARE** (RANZATO FRANCESCO) *1BC45* **BIOINFORMATICA** (VALLE GIORGIO) *1BC50*	**BIOINFORMATICA** (VALLE GIORGIO) *1BC50* **SISTEMI CONCORRENTI E DISTRIBUITI** (VARDANEGA TULLIO) *1BC45*
10:30 - 11:30	**LINGUAGGI DI PROGRAMMAZIONE AVANZATI** (CRAFA SILVIA) *1BC50*	**LINGUAGGI DI PROGRAMMAZIONE AVANZATI** (CRAFA SILVIA) *1BC50*	**SISTEMI CONCORRENTI E DISTRIBUITI** (VARDANEGA TULLIO) *1BC45* **SISTEMI CON VINCOLI** (LOMBARDI MICHELE) *LabTA*	**ANALISI STATICA E VERIFICA DEL SOFTWARE** (RANZATO FRANCESCO) *1BC45* **BIOINFORMATICA** (VALLE GIORGIO) *1BC50*	**BIOINFORMATICA** (VALLE GIORGIO) *1BC50* **SISTEMI CONCORRENTI E DISTRIBUITI** (VARDANEGA TULLIO) *1BC45*
11:30 - 12:30	**SISTEMI CON VINCOLI** (-- --) *1BC50* **SISTEMI CON VINCOLI** (LOMBARDI MICHELE) *LabTA* **SISTEMI CONCORRENTI E DISTRIBUITI** (VARDANEGA TULLIO) *1BC45*	**ANALISI STATICA E VERIFICA DEL SOFTWARE** (RANZATO FRANCESCO) *1BC50*	**SISTEMI CON VINCOLI** (LOMBARDI MICHELE) *LabTA* **SISTEMI CON VINCOLI** (-- --) *1BC50*	**INTELLIGENZA ARTIFICIALE** (SPERDUTI ALESSANDRO) *1BC50* **TECNOLOGIE WEB 2** (MARCHIORI MASSIMO) *1BC45*	**ANALISI STATICA E VERIFICA DEL SOFTWARE** (RANZATO FRANCESCO) *1BC50* **TECNOLOGIE WEB 2** (MARCHIORI MASSIMO) *1BC45*
	SISTEMI CON VINCOLI				

Problem Description

A number of lectures should be held in a number of rooms.

- Each lecture has a known number of participants
- Each room has a limited capacity that cannot be exceeded
- Lectures can be either long or short
- Long lectures are twice as long as short lectures
- In the considered period, there is time to host in each room:
  - A single long lecture
  - Or two short lectures

Problem Description

Build a (CP based) solution approach for the problem

- The file `lab02-timetabling.py` contains a template script
- The script accepts as a command line argument an instance file:

```
python lab02-timetabling.py <instance file>
```

- Instance files can be found in the folder `tt-data`
- During development, use `data-tt-debug.json`

Problem Description

Use the following constraint library as reference:

- Arithmetic expressions: `+, -, *, /, abs...`
- Equality/disequality/inequality `==, !=, <, <=, >=, >`

Try to:

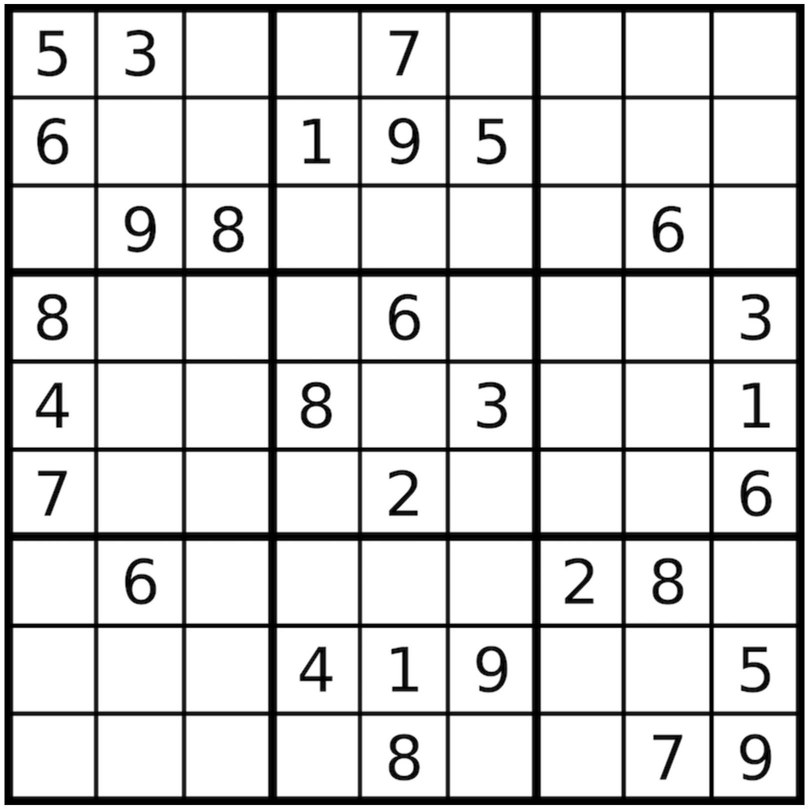
- Start without the constraint on short/long lectures
- Start to model with pen & paper! It helps at thinking

Constraint Systems

Lab 2 - Sudoku

Problem Description

Build a CP based solver for the popular Sudoku puzzle.



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Problem Description

Build a CP based solver for the popular Sudoku puzzle.

- The file `lab02-sudoku.py` contains a template script
- The script accepts as a command line argument an instance file

```
python lab02-sudoku.py <instance file>
```

- Instance files can be found in the folder `sudoku-data`
- During development, use `sudoku-easy-0.json`

Problem Description

Use the following constraint library as reference:

- Arithmetic expressions: `+, -, *, /, abs...`
- Equality/disequality/inequality `==, !=, <, <=, >=, >`

Then try to tackle the medium, hard, and "evil" instances

- Are they are as evil as they sound?
- How does the difficulty compare to the first problem?

Constraint Systems

Lab 2 - Tanks Problem

Problem Description

A number of chemicals must be stored in an array of tanks.

Problem Description

A number of chemicals must be stored in an array of tanks.

- There is a known amount of each chemical
- Each tank can store a single chemical
- Each chemical must be stored in a single tank
- Each tank has limited capacity, which cannot be exceeded
- Some chemicals are dangerous
  - Dangerous chemicals must be stored in special "safe" tanks
- Each chemical should be kept within a certain temperature range
  - If $tmax_i < tmin_j$ or $tmax_j < tmin_i$
  - Then chemical i and j cannot stay in adjacent tanks

Problem Description

Build a (CP based) solution approach for the problem

- The file `lab02-tanks.py` contains a template script
- The script accepts as a command line argument an instance file

```
python lab02-tanks.py <instance file>
```

- Instance files can be found in the folder `tank-data`
- During development, use `data-tanks-debug.json`
- Then attack more complex stuff

Problem Description

Use the following constraint library as reference:

- Arithmetic expressions: `+, -, *, /, abs...`
- Equality/disequality/inequality `==, !=, <, <=, >=, >`

Try to build the model incrementally

- And remember you don't have to start implementing immediately
- Plain old paper is good enough to design a model!