Constraint-based Modeling and Algorithms for Decision-making — INF555

Sylvain Soliman

September 18th, 2019
Part I

Decision problems, optimization, complexity and modelling
Decision Problems

- Finite input
  - Words
  - Rational numbers
  - Images
  - Sounds
  - Programs
  - ...

- yes/no output
  - providing an arbitrary solution is optional
  - providing all solutions is another class: enumeration problem
Examples

We will see during the class:

- Can we place $N$ queens on a chessboard with no attack?
- Do we have enough rooms for the lectures? assignment problem
- Can we land in the next 20 mn a given set of flights arriving in Orly? scheduling
- Can we find a sequence of actions to achieve a given goal? planning
- ...

Other classical examples include routing (traveling salesman), personnel staffing, etc.
Optimization Problems

- **Input:** finite data same as before

- **Output:** optimal cost
  - single number, or
  - vector of numbers for *multi-objective* optimization
  - providing an arbitrary optimal solution is optional
  - providing *all solutions* is another class: enumeration problem
• How many rooms are necessary to give the lectures?
• What is the time required to land a given set of flights arriving in Orly?
• How many flights from a given set of flights can land in the next 20min?
• ...

An optimization problem admits one associated decision problem: Is there a solution of given cost $k$?

Several optimization problems can be associated to a decision problem (choice of the inputs treated as objective function)
Mono-objective vs. Multi-objective optimization

The cost is always finite data

An *unique* rational number for mono-objective optimization.

- single optimization criterion
- possibly a robustness criterion w.r.t. perturbations
- the aggregation of multiple criteria representing a *trade-off*
Mono-objective vs. Multi-objective optimization

Some finite vector of rational numbers for multi-objective optimization

\[(f, g) < (f', g') \triangleq (f < f' \land g \leq g') \lor (f \leq f' \land g < g')\]

The optimal cost vectors \(\max(f, g)\) not unique: Pareto frontier
Computational Time-complexity Classes

An algorithm belongs to time-complexity class $C$ if it requires at most $C(n)$ elementary operations on a random access machine for an input of size $n$.

A problem belongs to class $C$ if there exists an algorithm in $C$ to solve it.

$$g(n) \in O(f(n)) \triangleq \exists k, n_0 \quad \forall n > n_0 \quad g(n) \leq k \cdot f(n)$$
## Examples

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(1)$</td>
<td>constant time, independent of the input</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>linear time</td>
</tr>
<tr>
<td>$O(n \log n)$</td>
<td>best possible complexity for sorting</td>
</tr>
<tr>
<td>$O(n^2)$</td>
<td>quadratic time</td>
</tr>
<tr>
<td>$P / PTIME$</td>
<td>polynomial time, i.e. $O(n^k)$ for some $k$</td>
</tr>
<tr>
<td>$O(2^n)$</td>
<td>polynomial time, i.e. $O(k^n)$ for some $k$</td>
</tr>
<tr>
<td>$EXPTIME$</td>
<td>exponential time, i.e. $O(2^{2^n})$</td>
</tr>
<tr>
<td>Non-elem.</td>
<td>not bounded by a finite tower of exponentials $O(2^{2^{\cdots}})$</td>
</tr>
</tbody>
</table>
Non-deterministic Time-complexity

**NP**: class of languages recognized in polynomial time by a *non-deterministic* Turing machine

≡ decision problems with proofs *verifiable in PTIME*

E.g., disjunctive scheduling, timetabling, planning, ...

**NP-complete**: class of problems in **NP** that can encode in polynomial time *any other NP problem*, i.e., the *hardest* NP problems

E.g., boolean satisfiability, graph coloring, disjunctive scheduling, ...
**NP-hard**: class of problems harder than NP, i.e., they can encode in polynomial time any NP problem.

E.g., *optimization problems* with NP-complete associated decision problem.
Space-complexity Classes

An algorithm belongs to a space complexity class \( C \) if it requires at most \( C \) memory locations.

PSPACE: polynomial space, i.e., \( O(n^k) \) for some \( k \)

Why do we have \( \text{NP} \subseteq \text{PSPACE} \)?
Space-complexity Classes

An algorithm belongs to a space complexity class $C$ if it requires at most $C$ memory locations.

PSPACE: polynomial space, i.e., $O(n^k)$ for some $k$.

Why do we have NP $\subseteq$ PSPACE?

Iterative deepening!

Bounded backtracking with increasing depth $d$.

At depth $d$, $d$ binary choices to remember, $O(d)$ space.

Computation of result at $O(n^k)$ depth: $O(n^k)$ space.
Time Complexity ≠ Size of the Search Space

Size of the solution domain:
bad upper bound for time-complexity

<table>
<thead>
<tr>
<th>Sorting $n$ integers</th>
<th>$!n$</th>
<th>$O(n \log n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Placing $n$ queens</td>
<td>$n^n$ or $!n$</td>
<td></td>
</tr>
</tbody>
</table>
Time Complexity ≠ Size of the Search Space

Size of the solution domain: bad upper bound for time-complexity

<table>
<thead>
<tr>
<th></th>
<th>!n</th>
<th>(O(n \log n))</th>
<th>(n^n) or !n</th>
<th>(O(1)) analytic solution</th>
<th>(\infty)</th>
<th>(O(1))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorting (n) integers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Placing (n) queens</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fermat Theorem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\forall n \exists abc\ a^n + b^n = c^n)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Time Complexity ≠ Size of the Search Space

Size of the solution domain: bad upper bound for time-complexity

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution Domain</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorting $n$ integers</td>
<td>$n!$</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>Placing $n$ queens</td>
<td>$n^n$ or $n!$</td>
<td>$O(1)$ analytic solution</td>
</tr>
<tr>
<td>Fermat Theorem</td>
<td>$\infty$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>$\forall n \exists abc \ a^n + b^n = c^n$</td>
<td>$n &gt; ? 2$ (A. Wiles 1994)</td>
<td></td>
</tr>
</tbody>
</table>

Separation results (like $P \neq NP$) are hard: need to quantify on all mathematical properties.
Theoretical Complexity only Bounds the Practical Complexity

**Worst-case complexity ≠ average time-complexity**  
(random inputs)

Can we solve NP-hard problems on very large instances?
Theoretical Complexity only Bounds the Practical Complexity

**Worst-case** complexity ≠ **average** time-complexity (random inputs)

Can we solve NP-hard problems on very large instances? **Yes!** But not on all (even small) instances (probably exponential)

Practical instances may happen to be easy
— *Polynomial class*
— *Phase transition*
— *Pathological examples* (exponential algorithm with polynomial empirical complexity)
Constraint Satisfaction Problems

Input:
- Finite set of Variables $x_1, \ldots, x_n$
- Corresponding Domains of values $D_1, \ldots, D_n$
- Finite set of Constraints $c_1, \ldots, c_k$
- Optional objective function $f(x_1, \ldots, x_n) \in \mathbb{R}$

Output:
- Decision $\exists? (x_1, \ldots, x_n) \in D_1 \times \cdots \times D_n$
  s.t. $c_1 \land \cdots \land c_k$
- Optimization $\min_{c_1 \land \cdots \land c_k} f(x_1, \ldots, x_n)$
- Solutions $\arg\min_{c_1 \land \cdots \land c_k} f(x_1, \ldots, x_n)$
The $N$ Queens Problem

$N$ Queens on an $N \times N$ chessboard with no attack
The $N$ Queens Problem

$N$ Queens on an $N \times N$ chessboard with no attack

Variables:
The $N$ Queens Problem

$N$ Queens on an $N \times N$ chessboard with no attack

Variables: $x_1, \ldots, x_N$ (columns)
The $N$ Queens Problem

$N$ Queens on an $N \times N$ chessboard with no attack

Variables: $x_1, \ldots, x_N$ (columns)
Domains:
The $N$ Queens Problem

$N$ Queens on an $N \times N$ chessboard with no attack

Variables: $x_1, \ldots, x_N$ (columns)
Domains: $D_i = \{1, \ldots, N\}$ (lines)
The $N$ Queens Problem

$N$ Queens on an $N \times N$ chessboard with no attack

**Variables:** $x_1, \ldots, x_N$ (columns)

**Domains:** $D_i = \{1, \ldots, N\}$ (lines)

**Constraints:**

- "not same line" $\forall i < j x_i \neq x_j$
- "not same diagonal" $\forall i < j x_i \neq x_j + i - j \land x_i \neq x_j - i + j$
The $N$ Queens Problem

$N$ Queens on an $N \times N$ chessboard with no attack

Variables: $x_1, \ldots, x_N$ (columns)
Domains: $D_i = \{1, \ldots, N\}$ (lines)
Constraints: “not same line”
The $N$ Queens Problem

$N$ Queens on an $N \times N$ chessboard with no attack

Variables: $x_1, \ldots, x_N$ (columns)
Domains: $D_i = \{1, \ldots, N\}$ (lines)
Constraints: “not same line”
\[ \forall i < j \ x_i \neq x_j \]
The $N$ Queens Problem

$N$ Queens on an $N \times N$ chessboard with no attack

Variables: $x_1, \ldots, x_N$ (columns)
Domains: $D_i = \{1, \ldots, N\}$ (lines)
Constraints: “not same line”
\[ \forall i < j \ x_i \neq x_j \]
“not same diagonal”
\[ \forall i < j \ x_i \neq x_j + i - j \land x_i \neq x_j - i + j \]
The $N$ Queens Problem

$N$ Queens on an $N \times N$ chessboard with no attack

Variables: $x_1, \ldots, x_N$ (columns)
Domains: $D_i = \{1, \ldots, N\}$ (lines)
Constraints: “not same line”
$\forall i < j \ x_i \neq x_j$
“not same diagonal”
$\forall i < j \ x_i \neq x_j + i - j \land x_i \neq x_j - i + j$
int: n;
array[1..n] of var 1..n: queens;

constraint forall (i, j in 1..n where i < j) (  
  queens[i] != queens[j] /
  queens[i] != queens[j] + j - i /
  queens[i] != queens[j] + i - j
);

solve satisfy;
(Declarative) Programming as Modeling

1940  Machine language
1954  Fortran: arithmetic expressions and control flow
1959  Lisp: functions over lists (Church’s $\lambda$-calculus)
1960  Algol: algorithms
1970  C: for whole operating system
1972  Prolog: first-order logic
1975  Smalltalk: objects
1978  ML: typed functions
1984  Constraint Logic Programming
1990  Constraint programming libraries for C++
1991  Python
1996  Java: object-oriented threaded programming
2008  Zinc: solver-independent constraint language
...

Von Neumann vs. Constraint machine

memory of values
programming variables

memory of constraints
mathematical variables

\[ V_i \leftarrow V_j + 1 \]

write

\[ X_i = X_j + 2 \]

add

\[ X_i \in [3,15] \]
\[ \sum a_i X_i \geq b \]
\[ \text{card}(1, [X \geq Y + 5, \ldots]) \]

read

and-concurrency

or-parallelism

\[ X_i \geq 5? \]
Constraints are Domain Filtering Agents

for each constraint $c_i$
for each variable $x_j$ in $c_i$
compute the projection $P_{ij}$ of the solutions on $x_j$
over-approximate if necessary
$D_j \leftarrow D_j \cap P_{ij}$

communication through shared variables in $\land$

Or-branches: no communication, easy to parallelize
And-concurrency: lots of comm., difficult to parallelize
Part II

MiniZinc, Jupyter and friends...
Most of the course will use MiniZinc as programming/modelling language.
A MiniZinc program may contain parameters

int: i = 1;
int j;
j = 2;

Fixed value (named constants), of type int, float, bool or string

May be given in a separate data file (.dzn)
MiniZinc syntax

A MiniZinc program may contain decision variables

```
var int: u;
var 1..10: v;
```

Only int or float, given with an optional domain

Parameters and variables can appear in constraints (using the usual arithmetic and Boolean relation operators)

```
constraint u = 18 * v + 42;
constraint alldifferent([x, y, z]);
```
Additional MiniZinc Syntax

It is possible to define sets and arrays of objects

```
set of int: STUDENT = 0..n;
% m[i] is the mark of student number i
array[STUDENT] of var int: m;
```

Iterators over those structures are given

```
constraint exists(s in STUDENT) (m[s] = 20);
constraint forall(s in STUDENT) (m[s] <= 20);
```
Final Bits

At least one `solve` statement must appear in a MiniZinc model

```plaintext
solve satisfy;
solve maximize u + 3*v;
solve minimize sum(i in STUDENT) (m[i]);
```

`output` takes a list of strings and displays them

Many other things (function or predicate definition, enums, comprehensions, etc.) ⟷ in TDs
Jupyter

Work will be done using **Jupyter Notebooks**

Assuming some basic knowledge of Python

Otherwise see: [https://docs.python.org/3/tutorial/introduction.html](https://docs.python.org/3/tutorial/introduction.html)

Code editing will use a version of **VS Code** embedded in a browser. (**vim** and **emacs** are also provided)
A MiniZinc model is composed of

- variable declarations
- constraints
- a `solve` statement

Let us first have a look at a partial model (file `aust.mzn`) for coloring Australia such that two neighbor states have a different color.

As you see, the basic editor is not that great... You can instead open a terminal (from the launcher) and launch vi/emacs if you like.

We will have colorized output in the notebook through a shell command

```
in [1]: !vimcat.sh aust.mzn

% Colouring Australia using nc colours
int: nc;

var 1..nc: wa;  var 1..nc: nt;  var 1..nc: sa;  var 1..nc: q;
var 1..nc: nsw; var 1..nc: v;  var 1..nc: t;

constraint wa != nt;
```
Generic module for using pymzn nicely in a Jupyter notebook.

Used in the class INF555

Written by Sylvain.Soliman@inria.fr

```python
import matplotlib.pyplot as plt
import numpy as np

from pymzn import Solver, cbc, chuffed, rebase_array,

from pymzn import minizinc as minizinc

def minizinc(mzn, *dzn_files, include=".", **kwargs):
    """Solve using minizinc."""
    return minizinc(mzn, *dzn_files, include=include, **kwargs)

class PicatSatSolver(Solver):
    """Solver instance for PicatSat."""

    def __init__(self):
        """Nothing special here."""
        super().__init__(solver_id='picat')

    def args(self, fzn_file, **kwargs):
        """Arguments for the CLI call."""
        return ['picat', 'fzn_picat_sat', fzn_file]
```
Docker

All will be run from Docker containers

Same environment for everyone

Do not forget to save your work! upload it to the Moodle at the end of the TD session (and later...)
Setup

1. Download Docker
   https://docs.docker.com/install/

2. Install Docker

3. Pull the image for the course
   
   ```bash
   docker pull \
   registry.gitlab.inria.fr/soliman/inf555
   ```
Windows Users

You might need Docker Toolbox (and not CE) if you are using *Family Edition*
https://docs.docker.com/toolbox/toolbox_install_windows/

You might need to use `192.168.99.100` instead of `localhost` for connecting to Jupyter

If all else fails, use VirtualBox to install an Ubuntu image and follow the instructions to install Docker there
TD1

Pull the missing files

docker pull \ registry.gitlab.inria.fr/soliman/inf555/td1

Run on local port 8888 with the work directory of the container pointing to where you launch the command

docker run -p 8888:8888 -p 8080:8080 -v \ "$PWD":/home/jovyan/work \ registry.gitlab.inria.fr/soliman/inf555/td1

You can now start the TD: http://localhost:8888/notebooks/TD.ipynb