A review of the Constraint Programming MOOC on EdX

Augustin Delecluse, Guillaume Derval, Laurent Michel, Pierre Schaus and Pascal Van Hentenryck

WTCP2023

Teaching in Constraint Programming

- Few universities proposes Constraint Programming (CP) courses
- Many discrete optimization courses focus on modeling
- Although modeling is important, there’s a need to teach CP mechanisms developed over the years

The MOOC

- Hosted on edX, a Massive Open Online Course (MOOC) provider
- MiniCP used as educational CP solver
- Content based on CP courses given at 4 universities, all using MiniCP
- Target audience: Master and PhD students, engineers, computer scientists
- Prerequisites
  - Familiarity with object-oriented programming
  - One algorithms course
  - Familiarity with git

Content of the course

Learning outcomes - Solvers

- Familiarity with the architecture of a CP solver
- Understanding advanced CP mechanisms
  - State restoration
  - Domain implementation
  - …
- Develop ability to implement Global Constraints and propagators
- Understand most popular black box search techniques
- Learn to implement backtracking search and search combinators (e.g. discrepancy search) within a solver

Learning outcomes - Modeling and Theory

- Engage with a wide range of combinatorial problems
  - Focus on vehicle routing and scheduling problems
- Develop skills to test, extend and improve existing code within CP models
- Understand balance and trade-offs between pruning strength and time complexity
  - Understand consistency (bound, domain, etc)
- Gain the ability to manipulate and employ the most frequently used constraints
  - Sum
  - Element
  - AllDifferent
  - …
- Understand mechanisms and application of reified constraints
- Learn to implement a problem specific search, variable and value heuristics

Teaching methodology

- 10 modules. Each module consists of:
  - Several videos (between 5 to 20 minutes)
  - MCQ about the videos (25% of grade)
  - Programming assignment (75% of grade)
- Features variety of speakers
- Focus first on key CP components
- Dives then into the most popular constraints





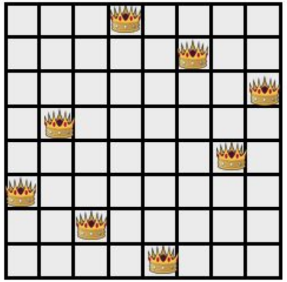

+ Valuable contributors to the teaching material: Pierre Flenner, Frej Knutar Lewander, Tias Guns, Christophe Lecoutre, Charles Prud'Homme, Peter Stuckey, Guido Tack

Table of content

Introduction

- Applications of CP in
  - Routing
  - scheduling
- CP as declarative paradigm
- N-Queens model

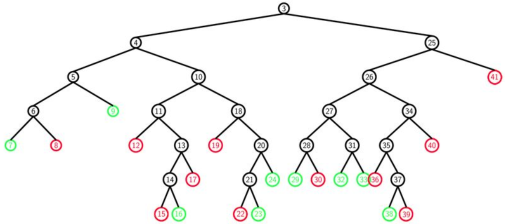
- *Model a graph coloring problem*



→ **MiniCP Solver**

- Domain implementation for Integer Variables
- Interfaces for variables and constraints
- Fix-Point algorithm
- DFS
- State Management

- *One constructor for Integer Variables*
- *Domain iterator*
- *Maximum constraint*



→ **Sum and Element**

- Domain and bound consistency
- Sum and Element constraints
- Reified constraints
- Quadratic Assignment Problem
- Stable Matching Problem

- *Several propagators for Element constraint*
- *Stable Matching implementation*

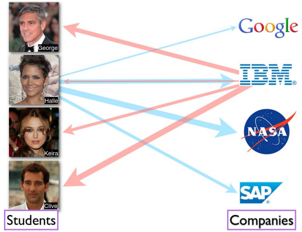


Table of content

- Usage of Table constraint
- Usage of bitsets
- Simple Tabular Reduction
- Compact Table

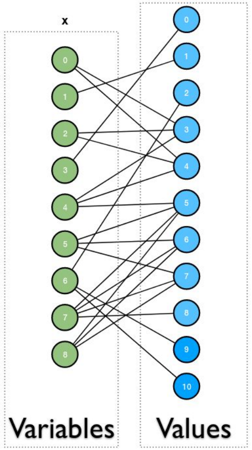
Table constraint

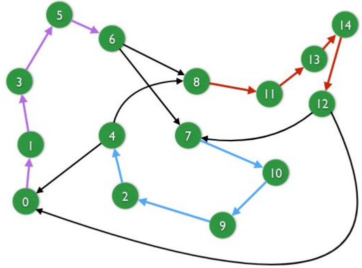
- *Implement Compact Table*
- *Model Eternity Problem*

- Forward checking for AllDifferent constraint
- Regin’s algorithm for domain consistency

All Different

- *AllDiff: Forward checking*
- *AllDiff: Regin’s algorithm*
- *Compare the implem on N-Queens*



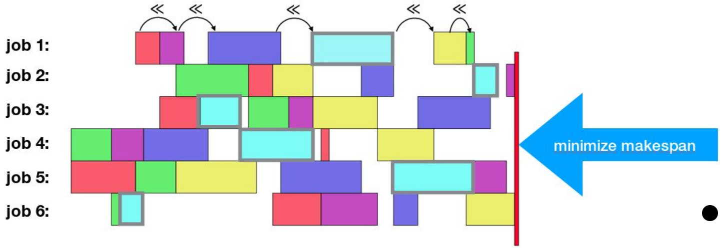


- Circuit constraint
- Modeling TSP
- Modeling VRP
- Large Neighborhood Search

TSP and Vehicle routing problems

- *Circuit constraint*
- *Custom search on TSP*
- *Parameter tuning for existing LNS*
- *Transform TSP into VRP*

Table of content



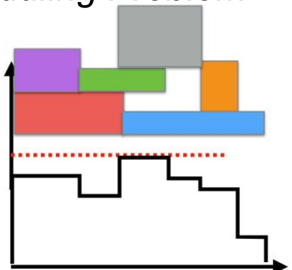
- Time-Tabling filtering
- LNS in scheduling
- Modeling producer-consumer
- Packing problems with cumulative

- Jobshop Problem
- Disjunctive constraint
- Theta-Tree datastructure

- First-Fail principle
- Impact search
- Activity-based search
- Conflict-based search
- Discrepancy search

Cumulative scheduling → Disjunctive scheduling → Black-Box Search

- *Cumulative decomposition*
- *Time-Tabling*
- *Resource-Constrained Project Scheduling Problem*



- *Modeling Jobshop*
- *Branching on the precedences for the Jobshop*
- *Implement missing filterings:*
  - *Detectable Precedence*
  - *Not-First/Not-Last*

- *Last conflict*
- *Conflict ordering*
- *Limited discrepancy search*

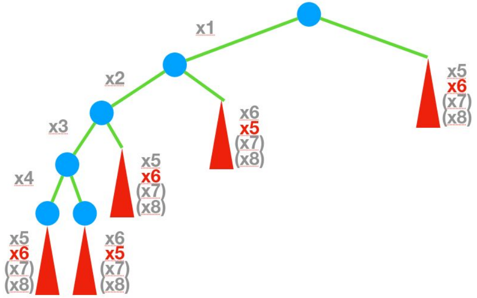


Table of content

- Bin-Packing
- Symmetry breaking
- Steel Mill Slab Problem

Modeling

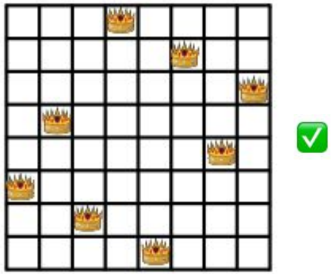
- *Or constraints with watched Literals*
- *Reified Or constraint*
- *Modeling Steel Mill Slab Problem and apply symmetry breaking on it*

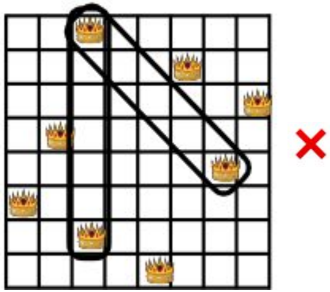
Real world

Models

Example with the first module

- 7 videos
  - Between 5 to 15 minutes each
  - Sum of duration: ~55 minutes
- Focus on N-Queens problem
  - Problem introduction
  - How to discover all solutions?
  - Gradually introduces CP components
- Clarification on declarative programming
- Examples of problems tackled with CP





- Applications of CP in
  - Routing
  - scheduling
- CP as declarative paradigm
- N-Queens model

Introduction

- *Model a graph coloring problem*

Programming assignment

Example with the first module

```java
/**
 * Solve the graph coloring problem
 * @param instance a graph coloring instance
 * @return the color of each node such that no two adjacent nodes receive a same color,
 *         or null if the problem is unfeasible
 */
public static int[] solve(GraphColoringInstance instance) {
    // TODO: solve the graph coloring problem using TinyCSP and return a solution
    // Hint: you can stop the search on first solution throwing and catching an exception
    //       in the onSolution closure or you can modify the dfs search
     throw new NotImplementedException("GraphColoringTinyCSP");
}
```

```java
/**
 * Solve the graph coloring problem
 * @param instance a graph coloring instance
 * @return the color of each node such that no two adjacent nodes receive a same color,
 *         or null if the problem is unfeasible
 */
public static int[] solve(GraphColoringInstance instance) {
    int n = instance.n;
    TinyCSP csp = new TinyCSP();
    Variable[] color = new Variable[n];
    for (int i = 0; i < n; i++)
        color[i] = csp.makeVariable(instance.maxColor);
    for (int [] edge: instance.edges) {
        int i = edge[0];
        int j = edge[1];
        // not the same color for adjacent nodes
        csp.notEqual(color[i],color[j], offset: 0);
    }
    ArrayList<int []> solutions = new ArrayList<>();
    // find the first solution
    try {
        csp.dfs(solution -> {
            solutions.add(solution);
            // stop the search at first solution
            throw new RuntimeException("stop");
        });
    } catch (RuntimeException stop) {
        return solutions.get(0);
    }
    return  null;
}
```

Variables

Constraints

Search

Programming assignment

Demo!

Programming assignment

- Grade obtained by passing unit tests
- Can be run
  - Locally - for self assessment and debugging
  - On a grading platform (INGInious) - validating the student’s score
- Creation, update and submission of assignment using git(hub)

Create your repository

Collapse context

The grading for this course will be partly based on your work on MiniCP.

You will do most of the work alone on a **private** repository hosted on GitHub. Please provide your GitHub username for its creation.

If you do not have a GitHub account yet, you can create one here.

Your answer passed the tests! Your score is 100.0%. [Submission #643ee855607332efce35ded7] ×

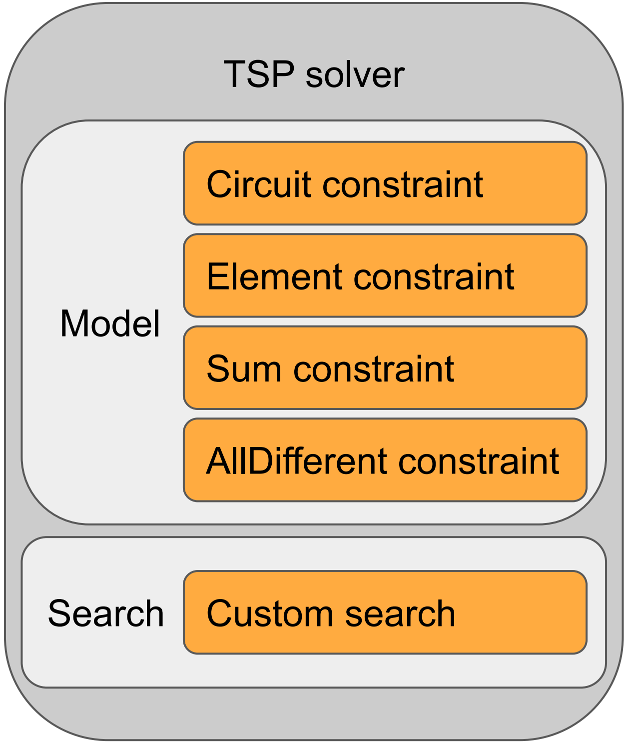
Your repository has been created! Click here to accept the invitation or look into your emails for it.

Your answer passed the tests! Your score is 100.0%. [Submission #643ee891607332efce35dedc] ×

Test	Status	Grade	Comment
GraphColoringTinyCSPTest	Success	1/1	
→ testSolve(String) - [1] gc_15_30_0	Success	0.1/0.1	
→ testSolve(String) - [2] gc_15_30_1	Success	0.1/0.1	
→ testSolve(String) - [3] gc_15_30_2	Success	0.1/0.1	

Categories of tests

- Smalls tests
  - Cover small and understandable examples
  - Useful for quick understanding and debugging
- Common mistakes tests
  - Based on common errors observed in previous years
  - Present mistakes in a comprehensible manners
- Runtime tests
  - Ensure correct usage of tips and datastructures
- Search tests
  - Check number of solutions / failures
  - More robust assessment of validity



A weird test?

Assignment 5 asks to implement a Circuit constraint propagator

Algorithm 1 Circuit propagation - beginning of the algorithm

Data: *dest*, *orig*: arrays of reversible integers storing the destination and origin of partial path through each Integer Variable x_i, respectively

Input : Integer Variable x_i that has become fixed

1. $j \leftarrow min(D(x_i))$;
2. $dest[orig[i]] \leftarrow dest[j]$;
3. ...

```
private void fix(int i) {
    // TODO
     throw new NotImplementedException("Circuit");
}
```

A weird test?

```java
private void fix(int i) {✅
    int j = x[i].min();
    int origi = orig[i].value();
    int destj = dest[j].value();
    dest[origi].setValue(destj);
```

```java
private void fix(int i) {❌
    int j = x[i].min();
    int origi = orig[i].value();
    dest[origi] = dest[j];
```

Algorithm 1 Circuit propagation - beginning of the algorithm

Data: *dest*, *orig*: arrays of reversible integers storing the destination and origin of partial path through each Integer Variable x_i, respectively

Input : Integer Variable x_i that has become fixed

$j \leftarrow min(D(x_i))$;

$dest[orig[i]] \leftarrow dest[j]$;

…

A weird test?

```java
private void fix(int i) { ✅
    int j = x[i].min();
    int origi = orig[i].value();
    int destj = dest[j].value();
    dest[origi].setValue(destj);
```

```java
private void fix(int i) { ❌
    int j = x[i].min();
    int origi = orig[i].value();
    dest[origi] = dest[j];
```

```java
// some people use dest[i] = dest[j] instead of calling setValue
// this compares the objects references to be sure that it is not the case
for (int i = 0 ; i < x.length; ++i) {
    StateInt origI = circuit.orig[i];
    for (int j = 0 ; j < x.length; ++j) {
        if (i != j) {
            assertNotSame(origI, circuit.orig[j],
                    message: "Use orig[i].setValue(...) to set the StateInt, not orig[i] = ...");
        }
        assertNotSame(origI, circuit.dest[j],
                message: "Use orig[i].setValue(...) to set the StateInt, not orig[i] = ...");
    }
}
```

A test that you would never find in non-educational CP solver!

Analytics

Analytics

- 515 enrolled students
- 110 attempted exercises

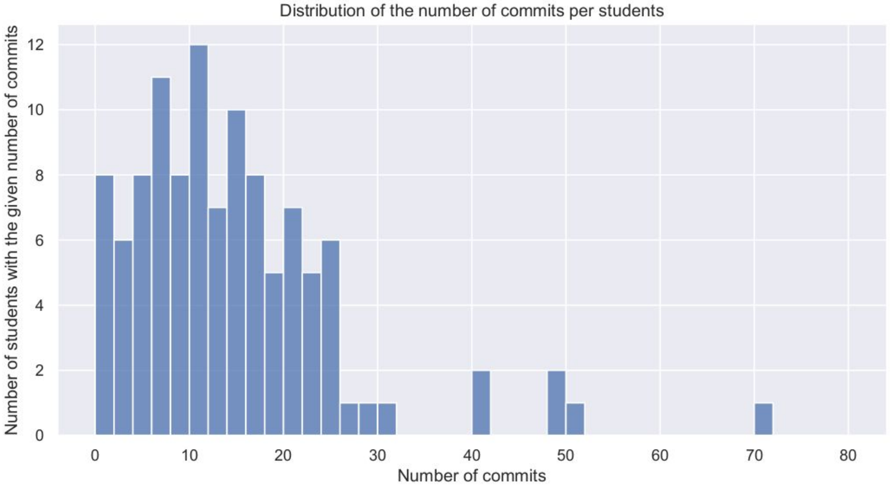


Figure 1 Distribution of the number of commits made by each student during the whole course. Each bin has a width of size 2 (0-1, 1-2, 3-4, ... are grouped together for readability).

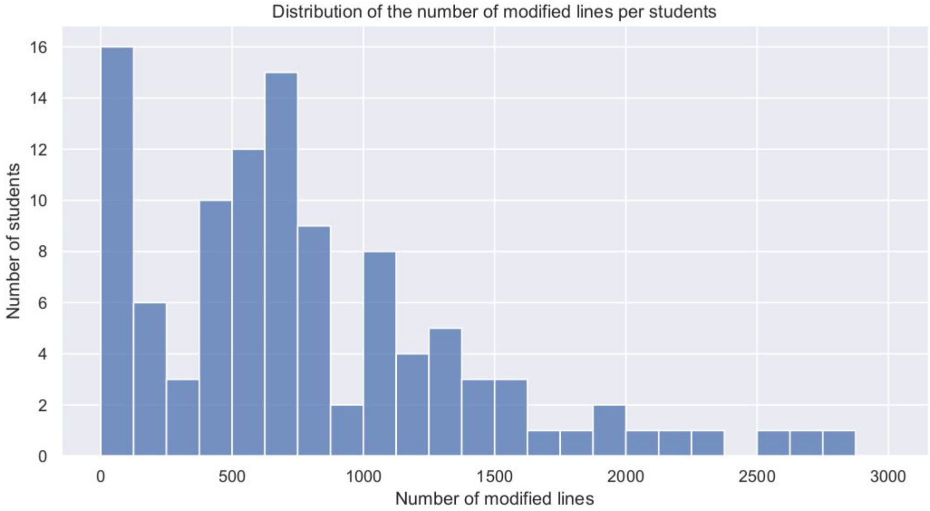


Figure 2 Distribution of the number of lines modified by each student during the whole course. Each bin has a width of size 250.

Analytics

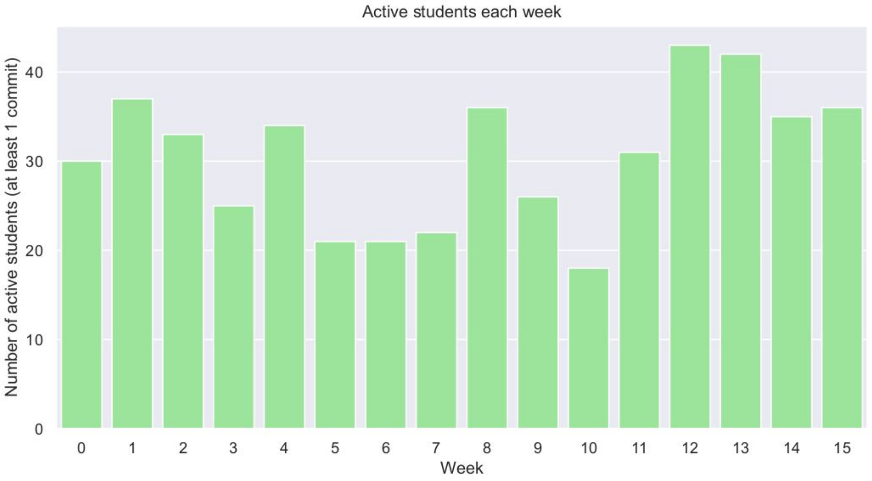


Figure 3 Number of active students per week. Active students are those who made at least one commit during a given week.

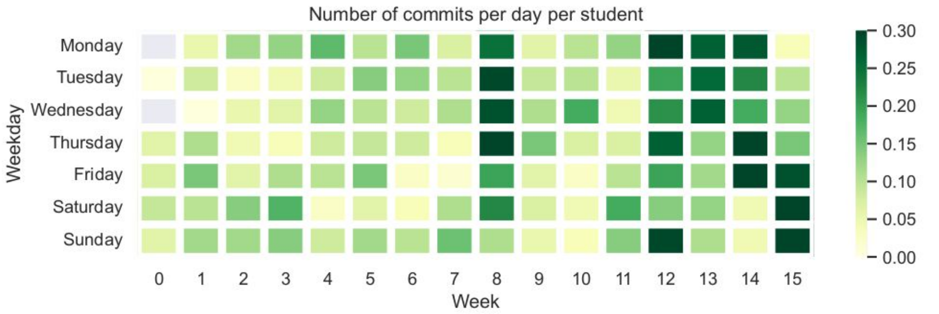


Figure 4 Average number of commits per day per student.

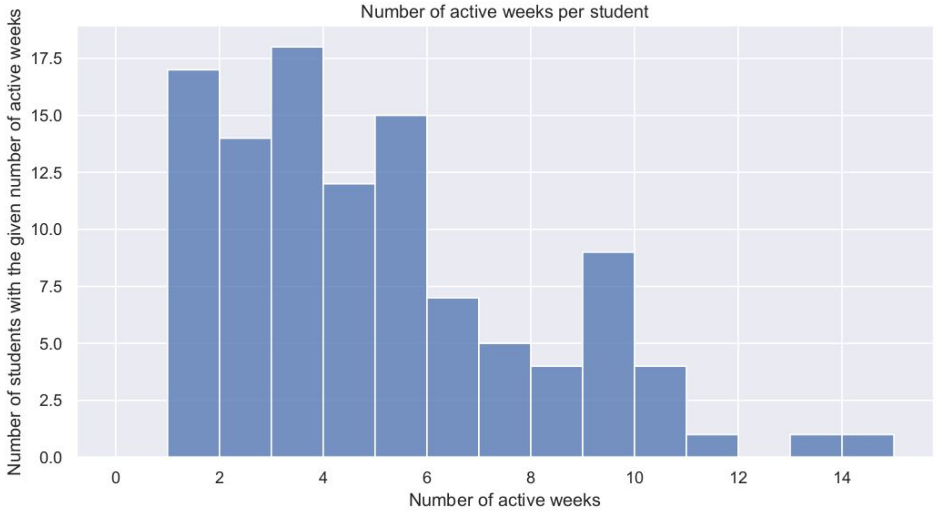
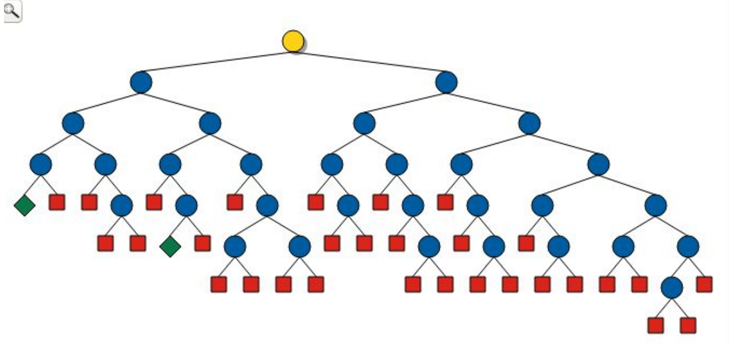


Figure 5 Number of "active weeks" per students. An active week is a week where the student made at least a commit.

Future of the MOOC

- Create exercise(s) where only a problem statement is given, and the students need to derive a working model and search for it
  - Currently the exercises are all guided
  - Evaluation framework can be easily adapted to such cases
- Give visualization tools to the students
- Invite CP expert for students with a stronger appetite for CP
- Next MOOC iteration starts on September 18th



Conclusion

- The MOOC provides a deep understanding of CP
  - Covers all key components of a CP solver
- Automation is the key to monitor and evaluate the students continuously and easily
- Room for improvement over next iterations
  - New exercises / tests
  - Visualisation tools
  - New experts highlights

https://edx.org/course/constraint-programming