Constraint Programming

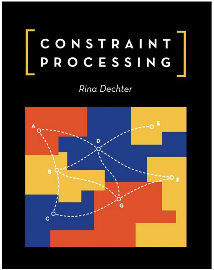
Matteo Zavatteri

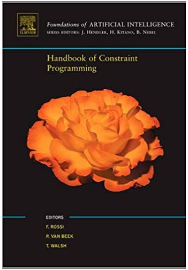
MathDecisions 2020-2021

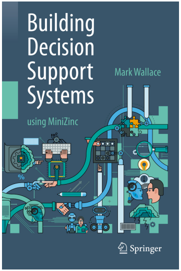
Outline

1. Constraint networks
2. Global constraints
3. More expressiveness
4. Modeling CSPs

Reference books







More online:
`http://www.constraint-programming.com/people/regin/papers/global.pdf`

`https://www.minizinc.org` (have a look at *MiniZinc Handbook*)

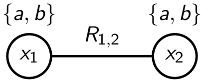
Constraint Networks

Formal definition

A *constraint network* is a triple $N = \langle X, D, C \rangle$, where:

1. $X = \{x_1, \ldots, x_n\}$ is a finite set of *(decision) variables*
2. $D = \{D_1, \ldots, D_n\}$ is a set of associated domains.
3. C is a finite set of *constraints*. Each constraint is a relation $R_{i,\ldots,k}$ (defined over the set of variables $\{x_i, \ldots, x_k\}$ such that $R_{i,\ldots,k} \subseteq D_i \times \cdots \times D_k$.

To ease notation, scopes and tuples are "ordered" with respect to variable indexes.



Formal specification

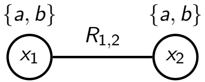
- $X = \{x_1, x_2\}$, $D = \{D_1, D_2\}$
- $D_1 = D_2 = \{a, b\}$
- $C = \{R_{1,2}\}$, where $R_{1,2} = \{(a, b), (b, a)\}$ (i.e., $x_1 \neq x_2$)

Consistency

Consistency

A constraint network is consistent if there exists a solution. That is, if every variable can be assigned a value from its domain such that all constraints are eventually satisfied.

Given a solution $x_1 = v_1, \ldots, x_n = v_n$, a constraint $R_{p,\ldots,z}$ is satisfied, if $(v_p, \ldots, v_z) \in R$, where $x_p = v_p, \ldots, x_z = v_z$ are in the solution.



Solution: $x_1 = a$, $x_2 = b$

A few problems associated to constraint networks

Given a constraint network N, we might address the following problems:

Decision problems:

- Is N consistent/inconsistent?
- Does N admit at least/at most/exactly k different solutions?
- Is there an assignment satisfying at least k constraints?

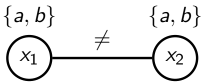
Search problems:

- Find a consistent assignment
- Find 2,3,etc different consistent assignments
- Find all consistent assignments
- How many different consistent assignments does N admit?
- Find an assignment maximizing the number of satisfied constraints

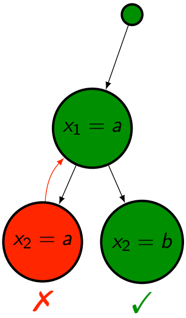
Search for 1 solution: backtracking

Assume a recursive algorithm that assigns variables according to the order of their indexes.

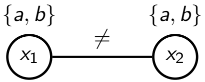
The algorithm stops *as soon as* it finds a solution



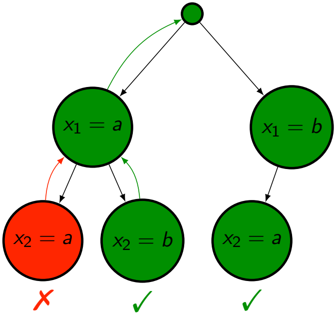
Solution 1: $x_1 = a$, $x_2 = b$



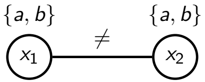
Search for 2 solutions: keep searching up to the 2nd



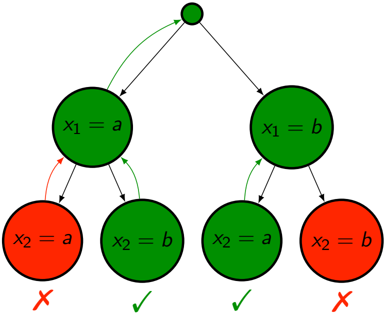
Solution 1: $x_1 = a$, $x_2 = b$
Solution 2: $x_1 = b$, $x_2 = a$



Search for all solutions: keep searching up to the end



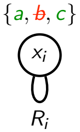
Solution 1: $x_1 = a$, $x_2 = b$
Solution 2: $x_1 = b$, $x_2 = a$



Filtering domains: node consistency

Node consistency

A variable x_i is node consistent if for each $v \in D_i$ we have that $(v) \in R_i$.



$R_i = \{(a), (c)\}$

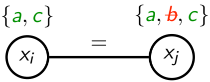
- x_i is not node consistent as $b \notin R_i$
- Removing b from D_i makes x_i node consistent

Rationale: *every solution must satisfy R_i and $x_i = b$ just doesn't.*

Filtering domains: arc consistency

Arc consistency

A pair of different variables x_i, x_j is arc consistent if for each $v_i \in D_i$ there exists $v_j \in D_j$ such that $(v_i, v_j) \in R_{ij}$.



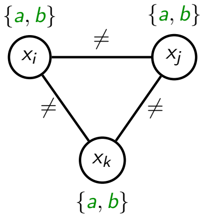
- x_i, x_j are not arc consistent. $(a, b) \notin R_{i,j}$, $(c, b) \notin R_{i,j}$
- Removing b from D_j makes x_i, x_j arc consistent.

Rationale: *every solution must satisfy $R_{i,j}$ and $x_j = b$ (whatever x_i) just doesn't.*

Filtering domains: path consistency

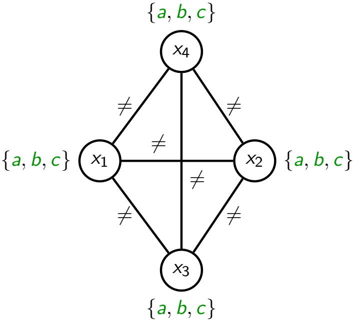
Path consistency

A pair of variables x_i, x_j is path consistent with another variable x_k ($x_i \neq x_j \neq x_k$) if for each $v_i \in D_i$, $v_j \in D_j$ with $(v_i, v_j) \in R_{ij}$, there exists $v_k \in D_k$ such that $(v_i, v_k) \in R_{ik}$ and $(v_j, v_k) \in R_{jk}$



- Arc consistent!
- Not path consistent. $x_i = a$, $x_j = b$ cannot be extended to any $x_k = v_k$ where $v_k \in \{a, b\}$.
- The network is actually inconsistent.

Path consistency is not enough!



- Path consistent!
- Yet, the network is actually inconsistent.
- All variables must get different values. Four variables. Three values.
- Examples like this extend to networks with n variables, $n-1$ values in each domain and a "$\neq$" constraint between any pair of distinct variables.
- Enforcing consistency on n variables says nothing for $n+1$ variables.

Node, arc and path consistency are *pruning techniques* to rule out (even many) values from domains. But eventually, we still need to search.

Global constraints

Take home message:
Global constraints = compact constraints

- they encapsulate several constraints in a single one
- they avoid writing an explicit relation of many, many tuples
- they typically involve several variables
- they allow for the specification of "high level" constraints

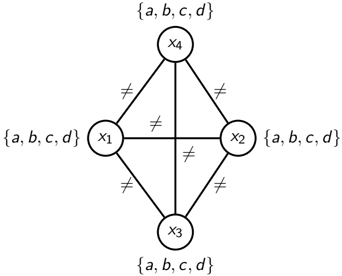
Examples:

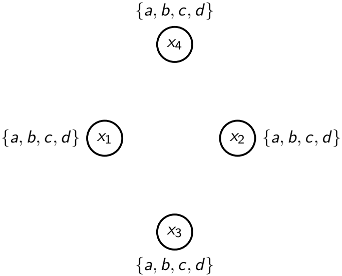
- $\texttt{all\_different}(x_1, \ldots, x_n)$
- $\ldots$

all_different

All different

A solution $x_1 = v_1, \ldots, x_n = v_n$ to a constraint network satisfies an `all_different`$(x_i, \ldots, x_j)$ iff $v_i \neq \ldots \neq v_j$.





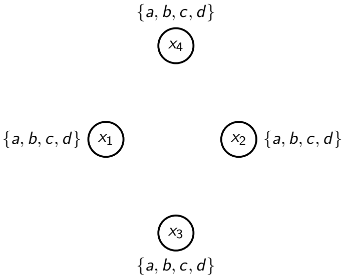
`all_different`(x_1, x_2, x_3, x_4)

$x_1 = a,\ x_2 = b,\ x_3 = c,\ x_4 = d$

all_different, more formally

A possible formal definition

$\texttt{all\_different}(x_1, \ldots, x_n)$ is equivalent to a relation $R_{1,\ldots,n}$ such that for each tuple $(v_1, \ldots, v_n) \in R$, it holds that $|\{v_i \mid v_i \in (v_1, \ldots, v_n)\}| = n$.



$\texttt{all\_different}(x_1, x_2, x_3, x_4)$

$$R_{1,2,3,4} = \{(a,b,c,d), (a,b,d,c), (a,c,b,d), (a,c,d,b), (a,d,b,c), (a,d,c,b), (b,a,c,d), (b,a,d,c), (b,c,a,d), (b,c,d,a), (b,d,a,c), (b,d,c,a), (c,a,b,d), (c,a,d,b), (c,b,a,d), (c,b,d,a), (c,d,a,b), (c,d,b,a), (d,a,b,c), (d,a,c,b), (d,b,a,c), (d,b,c,a), (d,c,a,b), (d,c,b,a)\}$$

$x_1 = c,\ x_2 = a,\ x_3 = b,\ x_4 = d$

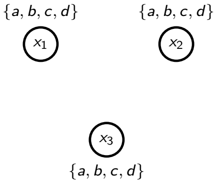
Boosting expressiveness maintaining complexity

Main complexity result

Deciding consistency of (classic) constraint networks is NP-complete.

- it is easy to see that the problem remains NP-complete even if we add global constraints or we turn a set of constraints into a boolean formula where global constraints and relations play the role of boolean atoms (provided that, given a solution, the satisfaction of each atom can be checked in polynomial time).

$F ::= R_{i,\dots,k} \mid \texttt{global\_constraint}(\dots) \mid \neg F \mid (F) \mid F \Box F$ where $\Box \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow, \dots\}$



Let $x_i = v_j$ be a short for $R_i = \{(v_j)\}$ (i.e., a further constraint language improvement!). The formula:

$$\texttt{all\_different}(x_1, x_2, x_3) \wedge (x_1 = a \vee x_3 = c) \wedge x_2 = d \wedge (x_1 = a \Rightarrow x_2 = c)$$

is satisfied by the solution $x_1 = b,\ x_2 = d,\ x_3 = c$

The solution (certificate of yes) can still be checked in polynomial time!

Humanizing relations by means of formulae

Consider the following constraint language:

$$F ::= x = v \mid (F) \mid F \wedge F \mid F \vee F$$

Wouldn't it be enough to encode a set of constraints $R_{i,\dots,z}$? (yes!)

Consider a constraint network $N = \langle X, D, C \rangle$ where:

- $X = \{x_1, x_2, x_3\}$
- $D_1 = D_2 = D_3 = \{a, b\}$
- $C = \{R_2, R_{13}, R_{123}\}$, where $R_2 = \{(b)\}$, $R_{13} = \{(a, a), (b, a), (b, b)\}$, $R_{123} = \{(a, b, a), (b, a, b)\}$

C can be encoded in a (DNF) formula $F \equiv \underbrace{F_2}_{R_2} \wedge \underbrace{F_{13}}_{R_{13}} \wedge \underbrace{F_{123}}_{R_{123}}$, where:

- $F_2 \equiv (x_2 = b)$
- $F_{13} \equiv ((x_1 = a \wedge x_3 = a) \vee (x_1 = b \wedge x_3 = a) \vee (x_1 = b \wedge x_3 = b))$
- $F_{123} \equiv ((x_1 = a \wedge x_2 = b \wedge x_3 = a) \vee (x_1 = b \wedge x_2 = a \wedge x_3 = b))$

In general $R_{i,\dots,z}$ can be encoded in $F \equiv (\bigvee_{(v_i,\dots,v_z) \in R_{i,\dots,z}} (x_i = v_i \wedge \dots \wedge x_z = v_z))$

Modeling constraint satisfaction problems (CSP)

In what follows, we will:

1. start with the definition of some problem in natural and formal language
2. model it in the input language of MiniZinc
3. push a button to search for one (or more) solution(s)

In this order. This is what we are going to do.

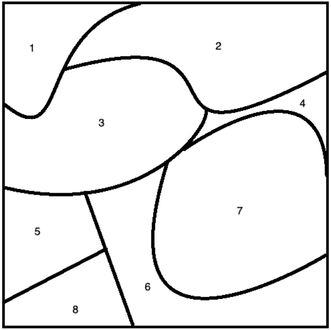
MiniZinc

`https://www.minizinc.org`

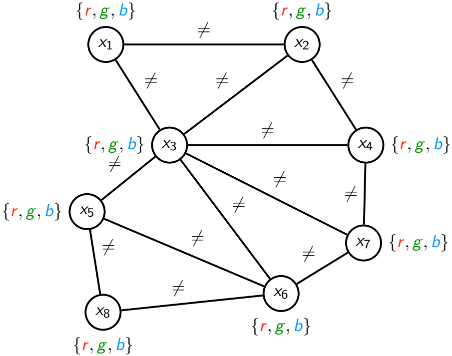
- `MiniZinc` is a free and open-source constraint modeling language that allows you to write models that are compiled into `FlatZinc`: an input language that is understood by a wide range of solvers.
- `MiniZinc` is developed at item Monash University `MonashUniversity` in collaboration with Data61 Decision Sciences `https://research.csiro.au/data61/tag/decision-sciences/` and the University of Melbourne `https://unimelb.edu.au`.
- `MiniZinc` is available for Windows, Linux and MacOS. Have a look at `https://www.minizinc.org/software.html`, download and install it on your computer.

Modeling CSPs: Map coloring

Can you color this map by using red, green and blue so that any two adjacent regions have different colors?

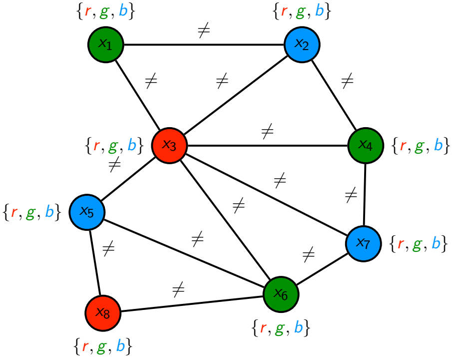


Constraint Network formulation

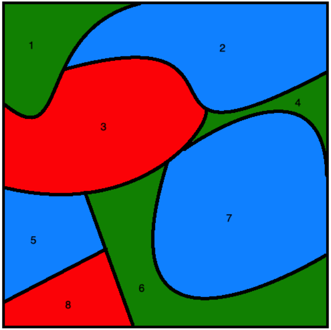


Modeling CSPs: Map coloring

Solution to the corresponding constraint network



Back to the map!

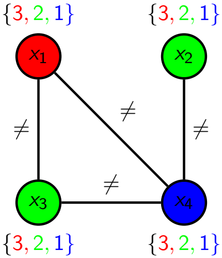


Graph K-Coloring Problem

Input. A graph $G = (V, E)$ and a positive integer K.
Output. yes iff there exists $f : V \to \{1, .., K\}$ s.t. $f(u) \neq f(v)$ for each $(u, v) \in E$

Example. $G = (V, E)$, where $V = \{x_1, x_2, x_3, x_4\}$ and

$E = \{(x_1, x_3), (x_1, x_4), (x_3, x_4), (x_2, x_4)\}$ and $K = 3$.



$f(x_1) = r$, $f(x_2) = g$, $f(x_3) = g$, $f(x_4) = b$.

Optimization version: Forget about K. Minimize the number of used colors.

Sudoku

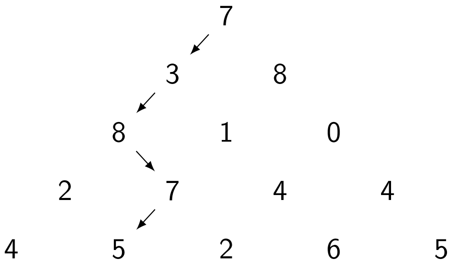
Input. A 9x9 grid in which each cell (i, j) must be filled with digits from 1 to 9. Some cells are prefilled (coherently with what the solution should look like).
Output. A filling of all empty cells of the grid such that each digit appears exaclty once in each row, each column, and each 3x3 subsquare.

Example.

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

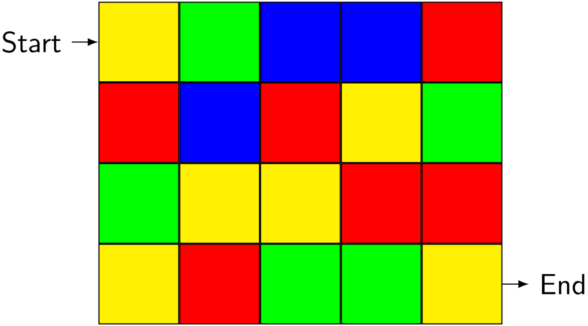
6	5	9	8	4	3	1	7	2
8	2	7	5	1	9	3	4	6
3	1	4	7	6	2	5	9	8
4	7	3	6	2	5	9	8	1
5	8	2	9	3	1	4	6	7
1	9	6	4	8	7	2	3	5
9	6	1	2	7	4	8	5	3
7	3	5	1	9	8	6	2	4
2	4	8	3	5	6	7	1	9

Triangle



Can you find a top-down path maximizing the overall sum?

Colored Grid Navigation



Find a walk from start to end.

- **At each step, you can either move Right or Down.**
- **You must visit: at least 1 red tile, at most 4 yellow tiles, exactly 1 blue tile, and no green tiles.**