Constraint Programming

Constraint Satisfaction Problem

Nicolas Barnier
nicolas.barnier@enac.fr

École Nationale de l'Aviation Civile

2018 – 2019

Objectives

- Understand the **Constraint Satisfaction Problem** (CSP) formalism and how it can represent **combinatorial optimization** problems
- Be able to enforce **arc** and **bound consistency** on a CSP
- Understand the **Branch & Prune** and **Branch & Bound** resolution algorithms
- Understand the importance and effects of **search strategies**

Introduction

Context

- Operations Research (OR): resource allocation, scheduling, vehicle routing (VRP $\supset$ TSP), configuration, bioinformatics, air traffic management (ATM)...
- Artificial Intelligence: SAT, partitioning, graphs (coloring, clique, covering)...

Non-linear, integers, disjunctive, arbitrary combinations...

Combinatorial Optimization

- Constraints : properties that a solution must verify
- Hard to build a valid solution ($\in$ NPC)
- Modeling formalism: CSP

Constraints

Restrict the combinations of values that can be assigned to variables

- Declarative
- Non-directional (non-functional $\neq$ spreadsheet)
- Additive: $\bigwedge_{c \in C} c$
- But disjunction possible: $c_1 \vee c_2$

Non-overlapping labels

$$\forall i \neq j, (x_i + l_i \leq x_j) \vee (x_j + l_j \leq x_i) \vee (y_i + h_i \leq y_j) \vee (y_j + h_j \leq y_i)$$

More Examples

Cryptarithmetic

$$
\begin{array}{cccccc}
 & & S & E & N & D \\
 & + & M & O & R & E \\
\hline
 & M & O & N & E & Y
\end{array}
$$

Sudoku

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

Outline

Constraint Satisfaction Problem

Definition (CSP)

A *CSP* (or *Constraint Network*) is defined by a triplet (X, D, C):

- $X = \{x_1, \ldots, x_n\}$ is the set of *variables* (i.e. unknowns).
- Each variable $x \in X$ is associated with its *domain* $d_x \in D$ of possible values.
- C is the set of *constraints*. Each constraint $c \in C$ is defined over a subset of variables $X_c \subseteq X$ by a relation $R_c \subset \prod_{\forall x \in X_c} d_x$ that specifies the set of allowed tuples (combinations) for X_c.

Solution

Definition (Solution of a CSP)

Assignment ϕ over all variables s.t. $\forall x \in X, \phi(x) \in d_x$ and all constraints are satisfied: $\forall c \in C, \phi(X_c) \in R_c$

Solving a CSP

- One solution
- All solutions / prove there is none
- Best solution w.r.t. a cost
- Over-constrained: minimize number of “conflicts”

Trivial Examples

Draws at tic-tac-toe

$$\begin{cases} X = \{\forall i,j \in [1,3]^2\ x_{i,j}\} \quad D = \left\{\{0,1\}^{3^2}\right\} \\ C = \left\{ \begin{array}{l} \forall i \in [1,3]\ (\{x_{i,1}, x_{i,2}, x_{i,3}\}, R), \\ \forall j \in [1,3]\ (\{x_{1,j}, x_{2,j}, x_{3,j}\}, R), \\ (\{x_{1,1}, x_{2,2}, x_{3,3}\}, R), \\ (\{x_{1,3}, x_{2,2}, x_{3,1}\}, R) \end{array} \right\} \end{cases}$$

with $R = \{(0,0,1),(0,1,0),(0,1,1),(1,0,0),(1,0,1),(1,1,0)\}$

Canadian flag

$$\begin{cases} X = \{l, c, r, m\} \\ D = \{\text{white}, \text{red}\}^4 \\ C = \left\{ \begin{array}{l} (\{l,c\}, l \neq c), \\ (\{c,r\}, c \neq r), \\ (\{c,m\}, c \neq m), \\ (\{m\}, \{\text{red}\}) \end{array} \right\} \end{cases}$$

Domains

CLP($\mathcal{X}$)

Generic formalism w.r.t. a mathematical structure $\mathcal{X}$:

- Finite trees (Prolog)
- Finite domains (integers)
- Rational, real (floating point) numbers
- Finite sets: $s \in [\emptyset, \{1,2,3\}]$, i.e. $\emptyset \subseteq s \subseteq \{1,2,3\}$
- “Mixed”: e.g. cardinal of a set $card = |s|$

Specification and Characteristics

Constraints

- In **extension**: allowed (or forbidden) tuples
- In **intention**: characteristic function
- Binary: $\forall c \in C, |X_c| = 2$
- No unique representation of a CSP

Characteristics

- **Dimension**: n
- **Size** of the search space: $\prod_{x \in X} |d_x|$
- Constraint **density** of a binary CSP: $\frac{|C|}{n(n-1)/2}$
- **Tightness** of a constraint: $1 - \frac{|R_c|}{|\prod_{\forall x \in X_c} d_x|}$

Constraints in Intention

Constraints in extension

Not very expressive/concise, costly algorithms, impossible in general for continuous domains

Constraints in intention with specific algorithms

- Arithmetic operators, inequalities and inequations: $+, -, \times, x^e, |x|, \mod ; =, <, \leq, \neq$
- Set operators, equations: $\cup, \cap, |s|, \subset, \in$...
- **Global** constraints: arithmetic ($\sum a_i x_i = d$) and symbolic (indexation, alldifferent, cardinality...)
- Logical operators, reification (metaconstraint): $\wedge, \vee, \Rightarrow$...

Constraints "language" $\longrightarrow$ modeling with CP solvers

Resolution of CSPs

Exhaustive Search

- Enumeration algorithms
- Constraint propagation: active use of the constraints
- Exponential complexity (worst-case): exploitation of the instance structure
- Completeness: proof (optimality, no solution)

Approximation algorithms

- Local search (LS), greedy algorithms
- Minimization of the number of violated constraints: "passive" use of the constraints, cost?
- Polynomial-time complexity?
- Not complete : "good solutions", large scale problems

Generate and Test

Definition (Local Consistency)

A **partial** assignment ϕ_V over a subset of variables $V \subseteq X$ is **locally consistent** iff it does not violate any constraint:

$$\forall c \in C \text{ t.q. } X_c \subseteq V, \phi_V(X_c) \in R_c$$

Backtracking (BT)

BT(V, ϕ) : bool

```
if V = ∅ then return true;
x ∈ V;
for a ∈ d_x do
    ϕ' ← ϕ ∪ {(x, a)};
    if ϕ' is locally consistent then
        if BT(V \ {x}, ϕ') then return true;
    end
end
return false;
```

Look-Ahead

Removal of inconsistent values *before* assignment

- BT not very efficient: constraints are checked **too late**
- **Look-ahead**: on variable assignment or domain restriction, values that cannot be part of a solution can be **inferred**
- Such values can be safely removed from domains: **filtering** that allows to **prune** the search tree
- Domain of each variable **memorized** before any filtering during search, to be **restored** in case of backtrack

Definition (Support for a value on a binary constraint)

A **support** for a value $a \in d_x$ of variable x on a binary constraint c with $X_c = \{x, y\}$ is a value $b \in d_y$ s.t. $(a, b) \in R_c$.

Justification to keep a value in its domain w.r.t. a constraint

Arc Consistency

Definition (Arc Consistency)

A binary constraint c with $X_c = \{x, y\}$ is arc-consistent iff all the values of d_x and d_y have a support on c.
A CSP is arc-consistent iff all its constraint are arc-consistent.

Filtering of d_x

Revise(x,y) : bool

```
modif ← false;
for a ∈ d_x do
    if a has no support in d_y on c then
        d_x ← d_x \ {a};
        modif ← true;
    end
end
return modif;
```

Enforcing Arc Consistency

An *arc-consistent* but not (globally) *consistent* CSP

$$(x, y, z) \in [1..2]^3,\ x \neq y \wedge y \neq z \wedge x \neq z$$

Constraint propagation [Mackworth 77]

AC-3(C)

$Q \leftarrow \{(x, y), (y, x),\ \forall c \in C\}$;
while $Q \neq \emptyset$ **do**
  $(x, y) \in Q$;
  $Q \leftarrow Q \setminus \{(x, y)\}$;
  if *Revise(x,y)* **then**
    $Q \leftarrow Q \cup \{(z, x),\ \forall c \in C \text{ s.t. } X_c = \{x, z\}, z \neq y\}$;
  end
end

AC-4 [Mohr 86], AC-6 [Bessière 93], GAC [Bessière 97]...

Branch & Prune

BT + filtering after each assignment/decision

- Arc consistency alone does not generally solve a CSP
- Interleaved with BT **search**: complete algorithm
- **Trade-off** between time spent during constraint propagation and the impact of pruning
  - Arc consistency: all constraints until fix point
  - Bound consistency: arithmetic constraints (interval arithmetic)
  - Cheaper: Forward Checking...
- When a domain is emptied by filtering: failure

Branch & Prune

Branch & Prune

- Inference: **deterministic** pruning (some level of local consistency enforced at each node)
  - Domain filtering
  - Inconsistency detection (e.g. alldifferent)
- Decision: if unknowns remain, **non-deterministic** choice
  - Assignment, domain splitting, constraint addition
  - In case of failure: **backtrack**

Maintaining Arc Consistency (MAC)

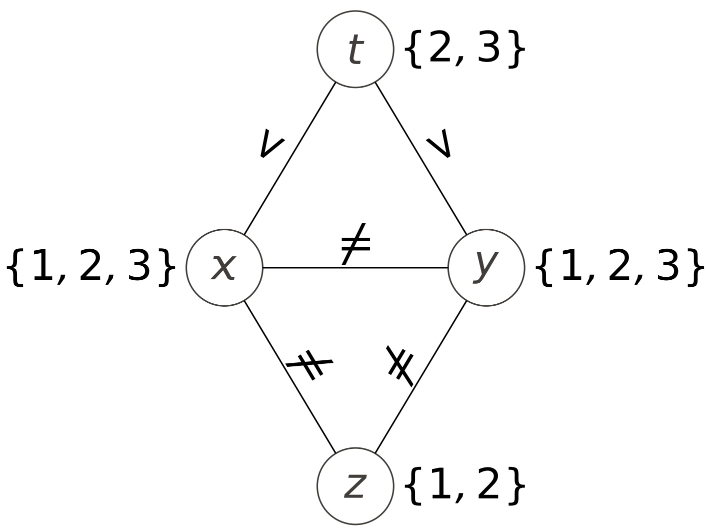
Incremental enforcement of arc consistency:

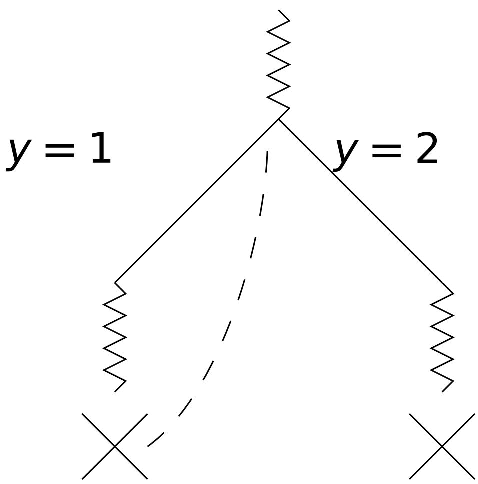
- Constraints checked only if their variables are modified
- Propagation events: assignment, bounds, domain
- Maintenance of internal data structures

Constraint Graph $(X, \{X_c, c \in C\})$

Constraint propagation during search

$$x, y \in \{1,2,3\} \qquad z \in \{1,2\} \qquad t \in \{2,3\}$$
$$x \neq y \qquad y \neq z \qquad z \neq x \qquad t > x \qquad t > y$$





Search Strategy

Choice points

- Selection of the next **variable** to be assigned:
  "first-fail" principle
- Selection of the **value** to assign:
  the most likely to lead to a (good) solution

Example:

$$
\begin{aligned}
X &= \{x, y, z, t\} \\
D &= \{[1, 2], [1, 2], [1, 2], \\
&\quad [1, 100]\} \\
C &= \{x \neq y, x \neq z, y \neq z\}
\end{aligned}
$$

Heuristics

Variable Ordering

- Static
- Dynamic: more efficient, robust (adaptive)
- Multiple criteria: *e.g.* (min-size, max-degree)
- Domain specific: *e.g.* critical object / resource
- Adaptive: *e.g. Weighted Degree*

Optimization

Characterization of solutions

- Several solutions in general: choice
- Preferences: resource consumption, distance...
- Cost function directly obtained from the decision variables:

$$c = f(x_1, \ldots, x_n) \quad \text{with} \quad f = \max, \sum, \text{card} \ldots$$

  or associated to each value: $c = f(c_1(x_1), \ldots, c_n(x_n))$
- First solution: upper bound UB (minimization)
- Branch & Bound: dynamic constraint $c <$ UB updated after each solution
- Optimality proof: no solution for $c < c^*$ (LB)
- Importance of (redundant) lower bound constraints
- Exploration of the search tree: DFS, LDS, DBS...

Choice of a (Complete) Resolution Algorithm

Summary

- Filtering power / inference
- Variable ordering
- Value selection
- Optimization (step-wise, dichotomic, %)
- Exploration of the search tree