Constraint programming: Theory, applications and teaching

Philippe Morignot

INRIA, IMARA Project-Team
October 25, 2013
Part I ---

Theory
Introduction

• **Constraint programming is one paradigm for modelling and solving combinatorial problems.**

• A combinatorial problem is a problem defined by entities maintaining relations, to which one combination (a solution) must be found.

• **Other algorithms:** Genetic algorithms, mixed integer programming, search algorithms in a state space (e.g., A*), simulated annealing, tabu search, ...

• One solution, several solutions, best solution, no solution.
Example (1/3)

• Sudoku:
Example (2/3)

• N-queen problem (here, N = 8):
Example (3/3)

- Cryptarithmetic:

\[
\begin{array}{c}
\text{SEND} \\
\text{+ MORE} \\
\hline
\text{MONEY}
\end{array}
\]

\[
\begin{array}{c}
\text{UN} \\
\text{+ DEUX} \\
\hline
\text{NEUF}
\end{array}
\]

\[
\begin{array}{c}
\text{UN} \\
\text{+ DEUX} \\
\hline
\text{NEUF}
\end{array}
\]

\[
\begin{array}{c}
\text{MONEY} \\
\text{+ DEUX} \\
\hline
\text{NEUF}
\end{array}
\]
Difficulty

• The number of combinations to consider might be gigantic for a real size problem.
  – **Example:** for the sudoku game, a coarse evaluation of the number of combinations is \((8!)^9 \approx 10^{41}\)
  – For small combinatorial problems, (almost) every algorithm works ...

• Enumerating all combinations would take an enormous time, even on the fastest computer.
  – Combinatorial explosion
  – In the worst case, the number of possible combinations to consider is an exponential function of the size of one dimension of the data.
Intuition

• Take into account the structure of a problem: decompose the problem into
  – Variables
    • Each variable has a finite domain (variables expressed in extension).
  – Relations among variables (constraints)
    • A constraint must always hold. It reduces the variables’ domains.
• A unique algorithm intelligently uses this model.
  – Heuristics.
Applications

- Assignment problems: assigning rooms to speakers while respecting preferences, ...
- Planning problems: optimal aerial traffic planning (assigning flight corridors to aircraft, optimizing crew rotations, ...) 
- Scheduling problems: scheduling tasks so that they finish as soon as possible while minimizing resource consumption, ...
- Optimizing the location of electronic components in circuits
- ...

P. Morignot - Softshake'13
Model

- **Variables with finite domain:**
  - Variables: $V_i$
  - Domains: $D_j = \{ v_1, v_2, ..., v_{nj} \}$
  - Forall $i$, $V_i \in D_i$

- **Constraints:**
  - For $k$, $C_k = (X_k, R_k)$ with:
    - $X_k = \{ V_{i1}, V_{i2}, ..., V_{ik} \}$ // The variables in $C_k$
    - $R_k \subseteq D_{i1} \times D_{i2} \times ... \times D_{ik}$ // Possible values of these variables, together with constraint $C_k$

- A solution to a CSP (Constraint Satisfaction Problem) is a total consistant assignment.
Constraints (1/2)

• A constraint can be expressed:
  – In extension: provide the possible values for variables
  – Arithmetically: $<$, $\leq$, $>$, $\geq$, $=$, $\neq$, $+$, $-$, $/$, $\ast$, ...
  – Logically: $\leftrightarrow$, $\Rightarrow$, $\Leftarrow$, OU, ET, NON, ...
  – AllDifferent($x_1$, $x_2$, ..., $x_n$), ...

• A constraint might be:
  – Hard: it must always hold
  – Soft: it might be violated but according to a criterion

• A constraint might be:
  – Unary. Example: $x \in [1, 5]$  
  – Binary. Example: $x < y$  
  – N-ary. Example: AllDifferent($V_1$, $V_2$, ..., $V_n$)
Constraints (2/2)

• Example of hard constraints:
  – $x \in [1, 5] \land y \in [1, 5] \land x < y$

  ![Diagram showing constraints](image)

• Example of soft constraints:
  – In a scheduling problem,
    
    $Y = \#(t_i < \text{deadline}_i)$ et maximize $Y$
Model / Example (1/5)

• Sudoku:
  – A variable is a cell in a grid
    • If a cell already includes a number, it appears as a constant
  – A domain is the integers between 1 and 9
  – Constraints:
    • All the variables in a grid are different
    • All the variables in a line are different
    • All the variables in a column are different
Model / Example (2/5)

• N-queen (here, N = 8) :
  – A pair of variables \((x_i, y_i)\) for each queen \(i\). The queen \(i\) is at column \(x_i\) and at row \(y_i\)
  – The domain of variable \(x_i\) is \([1, 8]\)
  – The domain of variable \(y_i\) is \([1, 8]\)
  – Constraints:
    \[
    \begin{align*}
    x_i & \neq x_j & \quad \text{// Different columns} \\
    y_i & \neq y_j & \quad \text{// Different rows} \\
    x_i + y_i & \neq x_j + y_j & \quad \text{// Different 1st diagonal} \\
    x_i - y_i & \neq x_j - y_j & \quad \text{// Different 2\textsuperscript{nd} diagonal}
    \end{align*}
    \]
Model / Example (3/5)

• N-queen (here, N = 8):
  – The variable $x_i$ is the raw of the $i^{th}$ column on which is the $i^{th}$ queen.
  – The domain of $x_i$ is [1, 8]
  – **Constraints:**
    The constraints on columns hold by construction
    
    $x_i \neq x_j$        // Different raws
    $x_i + i \neq x_j + j$ // Different 1$^{st}$ diagonals
    $x_i - i \neq x_j - j$ // Different 2$^{nd}$ diagonals
• N-queen (here, N = 8) :
  – The cells of the chessboard are indexed from 1 to 64.
  – The variable $x_i$ is the index of queen $i$.
  – Constraints:
    $$\frac{x_i}{8} \neq \frac{x_j}{8}$$ // Different columns
    $$x_i \% 8 \neq x_j \% 8$$ // Different raws
  Constraints on the 1\(^{\text{st}}\) diagonal
  Constraints on the 2\(^{\text{nd}}\) diagonal
Model / Example (5/5)

• **SEND + MORE = MONEY**

• **Variables:**
  - S, M ∈ [ 1, 9 ]
  - E, N, D, O, R, N, Y ∈ [ 0, 9 ]

• **Constraints:**
  
  \[
  \begin{align*}
  D + E & = Y + 10 \times R1 \\
  N + R + R1 & = E + 10 \times R2 \\
  E + O + R2 & = N + 10 \times R3 \\
  S + M + R3 & = O + 10 \times M \\
\end{align*}
  \]

  Addition carry variables : R1, R2, R3 ∈ {0, 1}
Backtracking

• Backtracking: chronological, backjumping, conflict-directed

P. Morignot - Softshake'13
Backjumping

Suppose that the instantiation order of variables is: A, B, C, E, D.
- A = red; B = green; C = blue; E = red
- No value for variable D. With a chronological backtracking, backtrack on E!
- The conflict set of D is \{A, B, C\}, so backjumping backjumps on C.
Heuristics

• A heuristics helps to make a choice.
  – Expressed from variables, domains and constraints.
  – Or can be based on the application domain of the CSP.
  – In C++: `int heuristics(Assignment* assignment, Csp* csp);`

• On variables: CHOOSE-UNASSIGNED-VARIABLE()
  – Static / dynamic.
  – Minimum remaining value / most constrained variable / fail-first: choose the variable with the smallest domain.
  – Choose the variable with the least number of constraints.
  – ...

• On values: SORT-VALUES-DOMAIN()
  – Static / dynamic
  – Choose the value which removes the least number of values to other variables.
  – ...
Filtering

• What does assigning a variable to a value imply for the other variables?

• FORWARD-CHECKING: each time a variable \( V_i \) is instantiated to a value, consider variables \( V_j \) connected to variable \( V_i \) by a constraint \( C_k \), and remove from variable \( V_j \) ‘s domain the values which are inconsistent with \( C_k \).
Filtering using Forward-Checking

Map coloring: set colors to regions A, B, C and D, while avoiding that two colors are adjacent. Possible colors are:

A and B are green or red
C is green, blue or red
D is blue or green
Filtering using Forward-Checking

• Model:

```
\begin{itemize}
  \item A \neq B \neq C \neq D
  \item \{\text{green, red}\}
  \item \{\text{green, blue, red}\}
  \item \{\text{blue, green}\}
\end{itemize}
```
Filtering using Forward-Checking

- Let us assume that B is set to the value RED.
Filtering using Forward-Checking

- Assignment of A to the value GREEN.
Filtering using Forward-Checking

- Assignment of C to the value BLUE.
Filtering using Forward-Checking

- Assignment of D to the value GREEN.
Filtering using Forward-Checking

- Solution!
Filtering using Forward-Checking

- But now let us assume that C is set to the value GREEN
Filtering using Forward-Checking

- Assignment of D to BLUE, and of A and B to RED.
Filtering using Forward-Checking

- Removal of RED from B’s domain.
Filtering using Forward-Checking

- The domain of B is empty: FAILURE!
  (and then backtracking is required...)
Structure of the problem

Tree

• Constraints are a tree.
  – For all variables pair \((V_i, V_j)\), \(V_i\) and \(V_j\) are linked by at most one path.
Structure of the problem
Going back to a tree: removing variables
Structure of the problem
Going back to a tree: macro-variables
Constraint propagation

k-consistency

• A CSP is \( k \)-consistent iff, for all subset of \((k - 1)\) variables, and for all assignment of these variables, a consistent value can always be assigned to the \( k^{th} \) variable.

• Example:
  – 1-consistency = node consistancy
    • Each individual variable is consistent.
  – 2-consistency = arc-consistency
  – 3-consistency = path consistancy
Algorithms

Arc consistency : AC-1

• bool REVISE \( (V_i, V_j) \)  // REMOVE-VALUES-DOMAIN
  
is-revised <- FALSE
  
  FORALL \( v_i \) in Domain\((V_i)\) DO
    
    IF there does not exist \( v_j \) in Domain\((V_j)\) such that \((v_i, v_j)\) is consistant
    THEN
      remove \( v_i \) from Domain\((V_i)\)
      
is-revised <- TRUE
    END-FORALL
  return is-revised

• Remove from variable \( V_i \) 's domain the values which are inconsistent with arc \((V_i, V_j)\) and return TRUE if at least one value has been removed (the arc has been revised).

• Time complexity : \( O(d^2) \) with \( d = \) domain cardinality.
Algorithms

Arc consistency: AC-1

• Procedure **AC-1**(\(csp\))
  
  \[
  Q \leftarrow \{ \text{ } (V_i, V_j) \text{ in } \text{Arcs}(csp) \text{ with } i \neq j \}\]

  REPEAT
  
  \[
  \text{change} \leftarrow \text{FALSE}
  \]

  FORALL (\(V_i, V_j\)) in \(Q\) DO

  \[
  \text{change} = \text{REVISE}(V_i, V_j) \text{ or } \text{change}
  \]

  UNTIL !\text{change}

• Complexity in time: \(O(n^2d^3)\)

• If only one arc revision occurs, all arcs will be revised at the next iteration.

• Whereas only arcs \((V_i, V_k)\) have been impacted by revising \(V_k\), since a value of \(V_i\) might have been removed.
Arc-consistency is not enough: path consistency is required.

Strong k-consistency.

Algorithms AC1 to AC8.


• **Tools**: OPL Studio from IBM (ex-ILOG)
  

  CHIP from COSYTEC, CHOCO from EMN, ECLIPSE from IC-PARC, CASPER from INRIA, SICSTUS Prolog, etc.

• **French Association for Constraint Programming**: [http://afpc.greyc.fr/web/](http://afpc.greyc.fr/web/)
Part II --- Applications
Application 1: airports (1/2)

- Aircraft traffic increases by 5% every year on average.
  - Each aircraft company states flight plans, which sums up to tasks having to be executed by on-ground employees at fixed time.
- In an airport, there is a fixed number of on-ground employees.
- It is difficult for the manager to daily build a planning (600 tasks, 150 employees).
- The manager wants to plan 6-months ahead, and might change his plans (weather conditions).
Applications 1 : airports (2)
Application 2 : airports (1/2)

- Strategic planning, not operational planning.
- An company’s flights are decomposed into packets.
- A packet has attributes, a terminal has attributes.
- Legal constraints (e.g., max # of departures/arrivals per year), usage constraints (e.g., shutdown, take-off track capacity).
- Assigning 500 flight packets of weekly flights on a dozen of Parisian terminals.
Application 2: airports (2/2)

- Assigning objects to containers
  - Under constraints
  - With a cost function

- Solving technique:
  - Mixed integer programming.
  - But the cost function includes a division ...
  - So constraint programming.
Application 3: Flight corridors (1/3)

- Destroying targets might be vital for a country.
- Unmanned combat aerial vehicles (UCAVs).
Application 2: Flight corridors (2/3)

Legend:
- SAM site
- Target
- Surveillance Radar site
- UCAV formation
- Command and Control
- Hostile Area
- FEBA
- Ingress Corridor
- Egress Corridor
- Alternate airport

- No take-off and landing
- No air-to air refuelling
- No store downloading area
- No air to air combat
- No target attack
Application 3 : Flight corridors (3/3)

- Flyable space is decomposed into flight corridors.
- Flight in package, except during attack phase.
- Simulate:
  - on-ground objects (e.g., targets, radars, missiles, base);
  - Aircraft sensors/effectors (e.g., communication, flight command);
  - Usage of flight in package (e.g., anti-collision, electromagnetic imprint).
- Compute a path through flight corridors. On-line re-planning.
- Compute the trajectory inside flight corridors.
Part III ---
Teaching
Teaching (1/3)

• Taught in 2012 and 2013 to ~40 M1 students of graduate school EPITA in Paris.
  – 2 x 3h classes ; 2 x 3h exercices ; 3h exam on computers.

• The problem is rather to model the problem than to solve it.
  – Difference between a modelling language and a programming language.
  – The students write equations (e.g., cryptarithmetics) and the solver finds a solution. Magic!

• Have a glimpse over combinatorial explosion.
  – Try the N-queen problem with N = 8, 50, 100, 1000, 10000.
Teaching (2/3)

• Problems for exercises: Cryparithmetics, N-queen, Sudoku, magic square, knapsack, magic numbers, chemistry factory.

• One problem for exams: «Un fermier doit passer la rivière dans une barque juste assez grande pour lui et son chien, ou lui et sa chèvre, ou lui et ses choux. Les choux seront mangés s'il les laisse seuls avec la chèvre, et la chèvre sera mangée s'il la laisse seule avec le chien. Comment faire passer tout ce monde sans dégâts ?»
Teaching (3/3)

Histogramme des notes 2012

Histogramme des notes 2013
Part IV ---
Demo with OPL Studio from IBM
THANK YOU!