Introduction to Constraint Programming

Willem-Jan van Hoeve

Tepper School of Business, Carnegie Mellon University

ACP Summer School on Theory and Practice of Constraint Programming
September 24-28, 2012, Wrocław, Poland
General introduction
• Successful applications
• Modeling
• Solving
• CP software

Basic concepts
• Search
• Constraint propagation
• Complexity
Constraint Programming Overview

- **Artificial Intelligence**
  - search
  - logical inference

- **Operations Research**
  - optimization
  - algorithms

- **Computer Science**
  - data structures
  - formal languages

- **Constraint Programming**
Evolution events of CP

- 1970s: Image processing applications in AI; Search + qualitative inference
- 1980s: Logic Programming (Prolog); Search + logical inference
- 1989: CHIP System; Constraint Logic Programming
- 1990s: Constraint Programming; Industrial Solvers (ILOG, Eclipse,...)
- 1994: Advanced inference for alldifferent and resource scheduling
- 2000s: Global constraints; integrated methods; modeling languages
- 2006: CISCO Systems acquires Eclipse CLP solver
- 2009: IBM acquires ILOG CP Solver & Cplex
Successful applications
Schedule of 1997/1998 ACC basketball league (9 teams)

- various complicated side constraints
- all 179 solutions were found in 24h using enumeration and integer linear programming [Nemhauser & Trick, 1998]
- all 179 solutions were found in less than a minute using constraint programming [Henz, 1999, 2001]
• Gate allocation at the Hong Kong International Airport
• System was implemented in only four months, and includes constraint programming technology (ILOG)
• Schedules ~900 flights a day (53 million passengers in 2011)
Port of Singapore

- One of the world’s largest container transshipment hubs
- Links shippers to a network of 200 shipping lines with connections to 600 ports in 123 countries
- Problem: Assign yard locations and loading plans under various operational and safety requirements
- Solution: Yard planning system, based on constraint programming (ILOG)
• Netherlands Railways has among the densest rail networks in the world, with 5,500 trains per day
• Constraint programming is one of the components in their railway planning software, which was used to design a new timetable from scratch (2009)
• Much more robust and effective schedule, and $75M additional annual profit
• INFORMS Edelman Award winner (2009)
CP Modeling and Solving

• CP allows a very flexible modeling language

• Virtually any expression over the variables is allowed
  – e.g., \( x^3(y^2 - z) \geq 25 + x^2 \cdot \max(x,y,z) \)

• CP models can be much more intuitive than, e.g., MIP or SAT models
  – close to natural language
• CP variable types include the classical:
  – binary, integer, continuous
• In addition, variables may take a value from any finite set
  – e.g., \( x \) in \{a,b,c,d,e\}
  – the set of possible values is called the domain of a variable
• Lastly, there exist special `structured’ variable types
  – set variables (take a set of elements as value)
  – activities or interval variables (for scheduling applications)
**CP Constraints – Examples**

- **Algebraic expressions:**
  \[ x^3(y^2 - z) \geq 25 + x^2 \cdot \max(x,y,z) \]

- **Extensional constraints (‘table’ constraints):**
  
  \((x,y,z) \text{ in MyTupleSet}\)

- **Variables as subscripts (‘element’ constraints):**
  
  \(y = \text{cost}[x]\) (here \(y\) and \(x\) are variables, ‘cost’ is an array of parameters)

- **Reasoning with meta-constraints:**
  
  \[ \sum_i (x_i > T_i) \leq 5 \]

- **Logical relations in which (meta-)constraints can be mixed:**
  
  \((x < y) \text{ OR } (y < z) \Rightarrow (c = \min(x,y))\)

- **Global constraints:**

  \(\text{Alldifferent}(x_1, x_2, ..., x_n)\)

  \(\text{DisjunctiveResource}(\text{[start}_1, ..., \text{start}_n], \text{[dur}_1, ..., \text{dur}_n], \text{[end}_1, ..., \text{end}_n])\)
We need to sequence a set of tasks on a machine
  
  – Each task \( i \) has a specific fixed processing time \( p_i \)
  
  – Each task can be started after its release date \( r_i \), and must be completed before its deadline \( d_i \)
  
  – Tasks cannot overlap in time

Time is represented as a discrete set of time points, say \( \{1, 2, \ldots, H\} \) (\( H \) stands for horizon)
MIP model

• Variables
  – Binary variable $x_{ij}$ represents whether task $i$ starts at time period $j$

• Constraints
  – Each task starts on exactly one time point
    \[ \sum_j x_{ij} = 1 \text{ for all tasks } i \]
  – Respect release date and deadline
    \[ j \cdot x_{ij} = 0 \text{ for all tasks } i \text{ and } (j < r_i) \text{ or } (j > d_i - p_i) \]
• Tasks cannot overlap
  – variant 1
    \[ \sum_i x_{ij} \leq 1 \text{ for all time points } j \]
    we also need to take processing times into account; this becomes messy
  – variant 2
    introduce binary variable \( b_{ik} \) representing whether task \( i \) comes before task \( k \)
    must be linked to \( x_{ij} \); this becomes messy
• **Variables**
  – Let $start_i$ represent the starting time of task $i$
    takes a value from domain \(\{1, 2, \ldots, H\}\)
  – This immediately ensures that each task starts at exactly one time point

• **Constraints**
  – Respect release date and deadline
    \[ r_i \leq start_i \leq d_i - p_i \]
CP model

• Tasks cannot overlap:
  for all tasks i and j
  
  \((start_i + p_i < start_j) \text{ OR } (start_j + p_j < start_i)\)

That’s it!

(See also Lombardi’s lectures on scheduling for more advanced models)
Benefits of CP model

- The number of CP variables is equal to the number of tasks, while the number of MIP variables depends also on the time granularity (for a horizon H, and n tasks, we have H*n binary variables $x_{ij}$)
- The sequencing constraints are quite messy in MIP, but straightforward and intuitive in CP
The traveling salesperson problem asks to find a closed tour on a given set of $n$ locations, with minimum total length.

Input: set of locations and distance $d_{ij}$ between two locations $i$ and $j$. 
• Classical model based on ‘assignment problem’
• Binary variable $x_{ij}$ represents whether the tour goes from $i$ to $j$
• Objective

$$\min \sum_{ij} d_{ij} x_{ij}$$

1 Alternative MIP models exist, for example the time-indexed formulation uses $y_{ijt} = 1$ if we traverse $(i,j)$ at step $t$
• Need to make sure that we leave and enter each location exactly once

\[ \sum_j x_{ij} = 1 \quad \text{for all } i \]

\[ \sum_i x_{ij} = 1 \quad \text{for all } j \]

• Constraints to remove all possible subtours:
  – there are exponentially many

MIP methodology therefore applies specialized solving methods for the TSP
Variable $x_i$ represents the $i$-th location that the tour visits (variable domain is $\{1, 2, \ldots, n\}$)

Objective

$$\min d_{x_n, x_1} + \sum_{i=1}^{n-1} d_{x_i, x_{i+1}}$$

variables can be used as subscripts!

Constraint

$\text{alldifferent}(x_1, x_2, \ldots, x_n)$

‘global’ constraint
The CP model needs only $n$ variables, while the MIP model needs $n^2$ variables (n is #locations).

The MIP model is of exponential size, while the CP model only needs one single constraint (and element contraints).

The CP model is perhaps more intuitive, as it is based directly on the problem structure: the ordering of the locations in the tour.

Note: The specialized MIP solving methods outperform CP on pure TSP. In presence of side constraints (e.g., time windows), CP is often much faster than MIP.
In general

- CP variables can be
  - discrete (i.e., integer valued)

- CP constraints can be
  - non-linear
  - non-differentiable
  - discontinuous

Hence, no traditional exact Operations Research technique (LP, NLP, MIP, etc) can solve these models
Basics of CP solving

• CP solving is based on intelligently enumerating all possible variable-value combinations
  – backtracking search
• At each search state, CP applies specific constraint propagation algorithms
• These propagation algorithms are applied to individual constraints, and their role is to limit the size of the search tree
Example: Graph coloring

Assign a color to each country.

Adjacent countries must have different colors.

Can we do this with at most four colors?
Smaller (8 variable) instance
CP Model

Variables and domains: $x_i$ in \{r,g,b,p\} for all i

Constraints: $x_i \neq x_j$ for all edges (i,j)
Constraint propagation: can we remove inconsistent domain values?
Search: guess a value for a variable (but be prepared to backtrack): \( x_2 = r \)
Propagate

Constraint propagation
Search: guess a value for a variable (but be prepared to backtrack): $x_5 = g$
Constraint propagation
Search: guess a value for a variable (but be prepared to backtrack): $x_4 = b$
Continuing search & propagate

Next search choice: $x_6 = b$
Continuing search & propagate

Next search choice: $x_8 = p$
Continuing search & propagate

Next search choice: $x_3 = g$
We found a solution!
Solution
The solution process of CP interleaves

- **Domain filtering**
  - remove inconsistent values from the domains of the variables, based on individual constraints

- **Constraint propagation**
  - propagate the filtered domains through the constraints, by re-evaluating them until there are no more changes in the domains

- **Search**
  - implicitly all possible variable-value combinations are enumerated, but the search tree is kept small due to the domain filtering and constraint propagation
Observations

- depth-first search (DFS)
  - linear memory requirements
  - applied in most CP systems
- search tree can be deep
  - considered 6 out of 8 variables
  - depth ‘controls’ the exponential explosion of the search

How can we make the search tree smaller in general?
Can we color the graph with 3 colors?

Search choice: $x_2 = r$
(by symmetry, no need to consider $x_2 = g, b$)
Search choice: \( x_5 = g \)
(be prepared to backtrack)
Search & propagate

... and propagate

$x_7$ has an empty domain: we need to backtrack
Search & propagate

Propagate...

\[ x_2 = r \]
\[ x_5 = g \]
\[ x_5 \neq g \]
Search & propagate
Search & propagate

...and propagate

$x_7$ has an empty domain: we are done
Recall example: first propagation

Can we do more propagation?

After $x_2 = r$ we are done.
Introduce global constraints

- We can increase the inference by adding more knowledge to the solver
  - in this case, group not-equal constraints that form a clique
  - use `alldifferent` constraints

\[ \text{alldifferent}(x_1, x_2, ..., x_n) := \bigwedge_{i<j} x_i \neq x_j \]

Model 1: \( x_1 \in \{g, b\}, x_4 \in \{g, b\}, x_8 \in \{r, g, b\} \)
\[ x_1 \neq x_4, x_1 \neq x_8, x_4 \neq x_8 \quad \text{no propagation} \]

Model 2: \( x_1 \in \{g, b\}, x_4 \in \{g, b\}, x_8 \in \{r, g, b\} \)
\[ \text{alldifferent}(x_1, x_4, x_8) \quad x_8 = r \]
Impact of global constraint propagation

• See graph.aimmspack

• Graph coloring problem; random instances

• Can set *alldifferent* propagation level from ‘low’ to ‘extended’
  – ‘low’: pairwise not-equal constraints
  – ‘extended’: best possible propagation
  – notice the difference in search tree size (search choices or failures) and solving time
Global Constraints Summary

• Examples
  – Alldifferent, Count, BinPacking, SequentialSchedule, ParallelSchedule, NetworkFlow, ...

• Global constraints represent combinatorial structure
  – can be viewed as the combination of elementary constraints
  – expressive building blocks for modeling applications
  – embed powerful algorithms from OR, Graph Theory, AI, CS, ...

• Essential for the successful application of CP
  – When modeling a problem, always try to identify possible global constraints that can be used
## Embedded Algorithms

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Structure/technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>alldifferent</td>
<td>bipartite matching [Régin, 1994]</td>
</tr>
<tr>
<td>cardinality</td>
<td>network flow [Régin, 1999, 2002]</td>
</tr>
<tr>
<td>knapsack</td>
<td>dynamic programming [Trick, 2003]</td>
</tr>
<tr>
<td>regular</td>
<td>directed acyclic graph [Pesant, 2004]</td>
</tr>
<tr>
<td>circuit</td>
<td>network flow [Genc Kaya &amp; Hooker, 2006]</td>
</tr>
<tr>
<td>weighted circuit</td>
<td>AP [Focacci et al, 1999], 1-Tree [Benchimol et al., 2012]</td>
</tr>
<tr>
<td>sequence</td>
<td>dedicated algorithm [vH et al., 2006, 2009] [Maher et al., 2009]</td>
</tr>
<tr>
<td>disjunctive/cumulative</td>
<td>dedicated algorithm [Nuijten 1994, Carlier et al., 1994] [Vilim, 2009]</td>
</tr>
<tr>
<td>inter-distance</td>
<td>dedicated algorithm [Quimper et al., 2006]</td>
</tr>
</tbody>
</table>

...  

The ‘global constraint catalog’ currently contains 354 constraints
http://www.emn.fr/z-info/sdemasse/gccat/
Commercial

- IBM ILOG CP Optimizer: C++ library; free for academic users
- Kalis (Artelys / Fico): C++ library
- SICSTUS-Prolog: Prolog-based library
- CHIP V5 (Cosytec): C++/Prolog-based library
- Comet (Dynadec): C++ library; free for academic users
- SCIP: C library; free for academic users

1 SCIP actually solves ‘constraint integer programs’ and combines MIP/LP solving with CP and SAT techniques
Non-commercial (in terms of licensing)

- Gecode: C++ library
- Eclipse: Prolog-based library
- Google OR-tools: C++ library
- Choco: Java library
- JaCoP: Java library
- Minion: black-box solver; specific problem input
- Mistral: C++ library
Commercial

- IBM ILOG OPL: CP Optimizer, CPLEX (LP/MIP/QP)
- Xpress Mosel: Kalis, Xpress (LP/MIP/QP/NLP)
- AIMMS: CP Optimizer, most commercial LP/MIP solvers, and NLP solvers
- Comet: Comet, lp_solve, SCIP

Non-commercial / Academic

- Zinc, MiniZinc (G12): Gecode, JaCoP
- Numberjack (python): Mistral, SCIP
- Tailor/Essence: Minion
Hands-on Session

• Tomorrow (Tuesday) there will be a hands-on session
  – application: vehicle routing with side constraints
  – integrated model: LP-based column generation with CP scheduling model

• We will use the AIMMS software for this
  – you can also use your favorite other CP system, but make sure it can link to an LP solver and has CP scheduling functionality
  – if you want to use AIMMS, please make sure to have it installed before the session starts (let me know if you have issues)
  – there will also be PCs in the lab running AIMMS
Get started with AIMMS

- One-hour tutorial (general introduction)
- Extensive documentation on CP functionality in the manuals under `help`
- Example CP models
  - sudoku
  - graph coloring
  - single machine scheduling
see:
http://www.andrew.cmu.edu/user/vanhoeve/summerschool
Outline

General introduction
• Successful applications
• Modeling
• Solving
• CP software

Basic concepts
• Search
• Constraint propagation
• Complexity
A Constraint Satisfaction Problem, or CSP, consists of

- set of variables $\mathcal{V}$,
- variable domains $\mathcal{D}(v)$ ($v \in \mathcal{V}$),
- and set of constraints $\mathcal{C}$ on the variables.

A solution to a CSP:

assign to each variable a single element from its domain such that all constraints are satisfied.

A constraint solver has to

- find a solution to a CSP,
- or prove that no solution exists.
Example: Graph Coloring

Coloring the nodes of the graph:
What is the minimum number of colors such that any two nodes connected by an edge have different colors?
Graph Coloring - Model

variables $\mathbf{v}$: A, B, C, D, E, F
domains $\mathbf{d}$: \{r, b, g, y, c, p\}
constraints $\mathbf{c}$: A ≠ B, A ≠ C, A ≠ E, A ≠ F, B ≠ C, B ≠ D, B ≠ F,
                    C ≠ D, C ≠ E, D ≠ E, D ≠ F, E ≠ F
# Graph Coloring – Simple Solver

<table>
<thead>
<tr>
<th>( \mathcal{V} )</th>
<th>( \mathcal{D} )</th>
<th>( \mathcal{E} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>{r, b, g, y, c, p}</td>
<td>A \approx B</td>
</tr>
<tr>
<td>B</td>
<td>{r, b, g, y, c, p}</td>
<td>A \approx C</td>
</tr>
<tr>
<td>C</td>
<td>{r, b, g, y, c, p}</td>
<td>A \approx E</td>
</tr>
<tr>
<td>D</td>
<td>{r, b, g, y, c, p}</td>
<td>A \approx F</td>
</tr>
<tr>
<td>E</td>
<td>{r, b, g, y, c, p}</td>
<td>B \approx C</td>
</tr>
<tr>
<td>F</td>
<td>{r, b, g, y, c, p}</td>
<td>B \approx D</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B \approx F</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C \approx D</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C \approx E</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D \approx E</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D \approx F</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E \approx F</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
A &= r, A \approx r \\
B &= r, B \approx r \\
C &= r, C \approx r \\
D &= r, D \approx r \\
E &= r, E \approx r \\
F &= r, F \approx r
\end{align*}
\]
**Graph Coloring – Simple Solver**

<table>
<thead>
<tr>
<th>$\mathcal{V}$</th>
<th>$\mathcal{D}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>{r}</td>
</tr>
<tr>
<td>B</td>
<td>{b, g, y, c, p}</td>
</tr>
<tr>
<td>C</td>
<td>{r, b, g, y, c, p}</td>
</tr>
<tr>
<td>D</td>
<td>{r, b, g, y, c, p}</td>
</tr>
<tr>
<td>E</td>
<td>{r, b, g, y, c, p}</td>
</tr>
<tr>
<td>F</td>
<td>{r, b, g, y, c, p}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\mathcal{E}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A \neq B</td>
</tr>
<tr>
<td>A \neq C</td>
</tr>
<tr>
<td>A \neq E</td>
</tr>
<tr>
<td>A \neq F</td>
</tr>
<tr>
<td>B \neq C</td>
</tr>
<tr>
<td>B \neq D</td>
</tr>
<tr>
<td>B \neq F</td>
</tr>
<tr>
<td>C \neq D</td>
</tr>
<tr>
<td>C \neq E</td>
</tr>
<tr>
<td>D \neq E</td>
</tr>
<tr>
<td>D \neq F</td>
</tr>
<tr>
<td>E \neq F</td>
</tr>
</tbody>
</table>

$6^4 = 1296$ failures
Graph Coloring – Simple Solver

\[
\begin{array}{|c|c|}
\hline
\text{V} & \text{D} \\
\hline
A & \{r\} \\
B & \{b\} \\
C & \{g\} \\
D & \{r\} \\
E & \{b\} \\
F & \{g\} \\
\hline
\end{array}
\]

in total 1736 failures
Graph Coloring – Constraint Solver

<table>
<thead>
<tr>
<th>V</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>{r}, b, g, y, c, p }</td>
</tr>
<tr>
<td>B</td>
<td>{r, b, g, y, c, p}</td>
</tr>
<tr>
<td>C</td>
<td>{r, b, g, y, c, p}</td>
</tr>
<tr>
<td>D</td>
<td>{r, b, g, y, c, p}</td>
</tr>
<tr>
<td>E</td>
<td>{r, b, g, y, c, p}</td>
</tr>
<tr>
<td>F</td>
<td>{r, b, g, y, c, p}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A ≠ B</td>
</tr>
<tr>
<td>A ≠ C</td>
</tr>
<tr>
<td>A ≠ E</td>
</tr>
<tr>
<td>A ≠ F</td>
</tr>
<tr>
<td>B ≠ C</td>
</tr>
<tr>
<td>B ≠ D</td>
</tr>
<tr>
<td>B ≠ F</td>
</tr>
<tr>
<td>C ≠ D</td>
</tr>
<tr>
<td>C ≠ E</td>
</tr>
<tr>
<td>D ≠ E</td>
</tr>
<tr>
<td>D ≠ F</td>
</tr>
<tr>
<td>E ≠ F</td>
</tr>
</tbody>
</table>

A = r  \quad A ≠ r
Graph Coloring – Constraint Solver

<table>
<thead>
<tr>
<th>( V )</th>
<th>( D )</th>
<th>( E )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>{r}</td>
<td>A ( \neq ) B</td>
</tr>
<tr>
<td>B</td>
<td>{b,g,y,c,p}</td>
<td>A ( \neq ) C</td>
</tr>
<tr>
<td>C</td>
<td>{b,g,y,c,p}</td>
<td>A ( \neq ) E</td>
</tr>
<tr>
<td>D</td>
<td>{r,b,g,y,c,p}</td>
<td>A ( \neq ) F</td>
</tr>
<tr>
<td>E</td>
<td>{b,g,y,c,p}</td>
<td>B ( \neq ) C</td>
</tr>
<tr>
<td>F</td>
<td>{b,g,y,c,p}</td>
<td>B ( \neq ) D</td>
</tr>
</tbody>
</table>

A = r

A \( \neq \) r

A

B

C

D

E

F
Graph Coloring – Constraint Solver

<table>
<thead>
<tr>
<th>( \mathcal{V} )</th>
<th>( \mathcal{D} )</th>
<th>( \mathcal{E} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>{r}</td>
<td>A \neq B</td>
</tr>
<tr>
<td>B</td>
<td>{b}</td>
<td>A \neq C</td>
</tr>
<tr>
<td>C</td>
<td>{g,y,c,p}</td>
<td>A \neq E</td>
</tr>
<tr>
<td>D</td>
<td>{r,g,y,c,p}</td>
<td>A \neq F</td>
</tr>
<tr>
<td>E</td>
<td>{b,g,y,c,p}</td>
<td>B \neq C</td>
</tr>
<tr>
<td>F</td>
<td>{g,y,c,p}</td>
<td>B \neq D</td>
</tr>
</tbody>
</table>

\[ B = b \quad B \neq b \]

\[ A = r \quad A \neq r \]
Graph Coloring – Constraint Solver

<table>
<thead>
<tr>
<th>$V$</th>
<th>$D$</th>
<th>$E$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>{r}</td>
<td>A$\neq$B</td>
</tr>
<tr>
<td>B</td>
<td>{b}</td>
<td>A$\neq$C</td>
</tr>
<tr>
<td>C</td>
<td>{g}</td>
<td>A$\neq$E</td>
</tr>
<tr>
<td>D</td>
<td>{r,y,c,p}</td>
<td>A$\neq$F</td>
</tr>
<tr>
<td>E</td>
<td>{b,y,c,p}</td>
<td>B$\neq$C</td>
</tr>
<tr>
<td>F</td>
<td>{g,y,c,p}</td>
<td>B$\neq$D</td>
</tr>
</tbody>
</table>

A = r
B = b
C = g

A$\neq$r
B$\neq$b
C$\neq$g
## Graph Coloring – Constraint Solver

### Variables and Domains

<table>
<thead>
<tr>
<th>$\mathcal{V}$</th>
<th>$\mathcal{D}$</th>
<th>$\mathcal{C}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>{r}</td>
<td>A\neq B</td>
</tr>
<tr>
<td>B</td>
<td>{b}</td>
<td>A\neq C</td>
</tr>
<tr>
<td>C</td>
<td>{g}</td>
<td>A\neq E</td>
</tr>
<tr>
<td>D</td>
<td>{r}</td>
<td>A\neq F</td>
</tr>
<tr>
<td>E</td>
<td>{b,y,c,p}</td>
<td>B\neq C</td>
</tr>
<tr>
<td>F</td>
<td>{g,y,c,p}</td>
<td>B\neq D</td>
</tr>
</tbody>
</table>

### Constraints

- A\neq r
- A\neq b
- A\neq g
- A\neq D
- A\neq E
- A\neq F
- B\neq D
- B\neq F
- C\neq D
- C\neq E
- D\neq D
- D\neq F
- E\neq F
Graph Coloring – Constraint Solver

<table>
<thead>
<tr>
<th>V</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>{r}</td>
<td>A≠B</td>
</tr>
<tr>
<td>B</td>
<td>{b}</td>
<td>A≠C</td>
</tr>
<tr>
<td>C</td>
<td>{g}</td>
<td>A≠E</td>
</tr>
<tr>
<td>D</td>
<td>{r}</td>
<td>A≠F</td>
</tr>
<tr>
<td>E</td>
<td>{b}</td>
<td>B≠C</td>
</tr>
<tr>
<td>F</td>
<td>{g,y,c,p}</td>
<td>B≠D, B≠F, C≠D, C≠E, D≠E, D≠F, E≠F</td>
</tr>
</tbody>
</table>

Diagram:

- A = r, A ≠ r
- B = b, B ≠ b
- C = g, C ≠ g
- D = r, D ≠ r
- E = b, E ≠ b
- F

- A → B → C → D → E → F
Graph Coloring – Constraint Solver

<table>
<thead>
<tr>
<th>$v$</th>
<th>$D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>{r}</td>
</tr>
<tr>
<td>B</td>
<td>{b}</td>
</tr>
<tr>
<td>C</td>
<td>{g}</td>
</tr>
<tr>
<td>D</td>
<td>{r}</td>
</tr>
<tr>
<td>E</td>
<td>{b}</td>
</tr>
<tr>
<td>F</td>
<td>{g}</td>
</tr>
</tbody>
</table>

$c$

- $A \neq B$
- $A \neq C$
- $A \neq E$
- $A \neq F$
- $B \neq C$
- $B \neq D$
- $B \neq F$
- $C \neq D$
- $C \neq E$
- $D \neq E$
- $D \neq F$
- $E \neq F$

A = $r$  A $\neq r$
B = $b$  B $\neq b$
C = $g$  C $\neq g$
D = $r$  D $\neq r$
E = $b$  E $\neq b$
F = $g$  F $\neq g$

In total 0 failures!
A constraint solver interleaves search and constraint propagation.

- **Search**:
  1. **A**
  2. **B**
  3. **C**

- **Constraint Propagation**:
  1. $A \neq B \rightarrow$ remove $r$ from $\mathcal{D}(B)$
  2. $B \neq C \rightarrow$ remove $b$ from $\mathcal{D}(C)$

- **Exponential Size in General**
- **Reduce Size of Search Tree**
A search tree can be defined by

- **enumeration**
  - choose a variable \( v \) from \( V \)
  - choose an element \( e \) from \( D(v) \)
  - branch \( v = e \) versus \( v \neq e \)

- **partitioning**
  - choose a variable \( v \) from \( V \)
  - choose a subset \( S \) of elements from \( D(v) \)
  - branch \( v \in S \) versus \( v \notin S \)

- **branching constraints**
  - for example \( x \leq y \) versus \( x > y \)

Solving process heavily dependent on variable (and value) selection heuristics
Propagation

Constraint propagation makes the domains consistent with each individual constraint:

**Definition**: a constraint \( C \) is **domain consistent** if all elements of all variable domains belong to a solution to \( C \).

- Domain consistent example:
  - \( V, D \):
    - \( x \in \{0,1,2\} \)
    - \( y \in \{1,2,3\} \)
  - \( e \):
    - \( x < y \)

- Not domain consistent example:
  - \( V, D \):
    - \( x \in \{0,1,2,3,4\} \)
    - \( y \in \{0,1,2,3\} \)
  - \( e \):
    - \( x < y \)

also known as ‘arc consistent’
Q: How can we make a constraint domain consistent?
A: Reduce the domains of the variables in the constraint.

Such a procedure is called a *domain filtering algorithm* or ‘propagation’ algorithm.
A basic propagation algorithm for a constraint $C$ looks as follows:

```c
bool Propagate(C, V_C, D) {
    for all $v \in V_C$ {
        for all $e \in D(v)$ {
            find a solution to $C$ with $v=e$;
            if no solution exists, remove $e$ from $D(v)$;
            if $D(v)$ is empty return false;
        }
    }
    return true;
}
```

<table>
<thead>
<tr>
<th>$v$</th>
<th>$D$</th>
<th>$e$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>${0,1,2,3,4}$</td>
<td>$x &lt; y$</td>
</tr>
<tr>
<td>$y$</td>
<td>${0,1,2,3}$</td>
<td></td>
</tr>
</tbody>
</table>
A CSP usually consists of several constraints. What about constraint interaction?

**Definition:** A CSP is domain consistent if all its constraints are.

<table>
<thead>
<tr>
<th>C</th>
<th>(1)</th>
<th>(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x+y ≤ 4</td>
<td></td>
<td>all different(x,y,z)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>V</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>{2,3}</td>
</tr>
<tr>
<td>y</td>
<td>{1,2,3,4}</td>
</tr>
<tr>
<td>z</td>
<td>{1,2}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>V</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>{2,3}</td>
</tr>
<tr>
<td>y</td>
<td>{1,2}</td>
</tr>
<tr>
<td>z</td>
<td>{1,2}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>V</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>{3}</td>
</tr>
<tr>
<td>y</td>
<td>{1,2}</td>
</tr>
<tr>
<td>z</td>
<td>{1,2}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>V</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>{3}</td>
</tr>
<tr>
<td>y</td>
<td>{1}</td>
</tr>
<tr>
<td>z</td>
<td>{1,2}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>V</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>{3}</td>
</tr>
<tr>
<td>y</td>
<td>{1}</td>
</tr>
<tr>
<td>z</td>
<td>{2}</td>
</tr>
</tbody>
</table>
More formally, a CSP is made domain consistent by a

Propagation Cycle:

```cpp
bool Propagation_Cycle(V, D, C) {
    Q := V;
    while Q nonempty {
        pick v in V;
        Q := Q - v;
        for all constraints C in C containing v {
            if (Propagate(C, V, D) == false) return false;
            if a domain of variable x has changed, set Q := Q + x;
        }
    }
    return true;
}
```

Alternative: store active constraints in Q

Note: this process always reaches a fix-point (i.e. either domain consistency or a failure).
Solving process

Search + Constraint Propagation:
Solving process

Search + Constraint Propagation:

```c
bool Solve(V,D,C){
    if |D(v)| == 1 for all v in V {
        return true; // we found a solution!
    }
    else {
        if (Propagation_Cycle(V,D,C) == false) {
            return false;
        }
        else {
            choose variable v in V with |D(v)| > 1;
            choose element e in D(v);
            return ( Solve(V,D,C +‘v=e’) OR Solve(V,D,C +‘v≠e’) );
        }
    }
}
```
Consider binary constraint $x < y$, given as a ‘table’:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$f$</td>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
</tr>
<tr>
<td>1</td>
<td>$f$</td>
<td>$f$</td>
<td>$t$</td>
<td>$t$</td>
</tr>
<tr>
<td>2</td>
<td>$f$</td>
<td>$f$</td>
<td>$f$</td>
<td>$t$</td>
</tr>
<tr>
<td>3</td>
<td>$f$</td>
<td>$f$</td>
<td>$f$</td>
<td>$f$</td>
</tr>
<tr>
<td>4</td>
<td>$f$</td>
<td>$f$</td>
<td>$f$</td>
<td>$f$</td>
</tr>
</tbody>
</table>

$t = \text{true} \\
 f = \text{false}$

Example:

$x \in \{0, 1, 2, 3, 4\}$

$y \in \{0, 1, 2, 3\}$

$x < y$

check all elements in $D(x)$ for support
check all elements in $D(y)$ for support

this takes $O(|D(x)| \cdot |D(y)|)$ time.

This is the general time complexity to make binary constraints domain consistent.
More efficient propagation

Consider binary constraint \( x < y \) again.

Example: \( x \in \{0,1,2,3,4\} \)

\[ y \in \{0,1,2,3\} \]

\[ x < y \]

Note that this constraint is domain consistent iff

\[ \min(x) < \min(y) \text{ and } \max(x) < \max(y) \]

So we can make the constraint domain consistent by:

- remove from \( D(y) \) all values \( \leq \min(x) \)
- remove from \( D(x) \) all values \( \geq \max(y) \)

This takes \( O(1) \) time (update the bounds of the domains).

Much more efficient than generic algorithm.
What about non-binary constraints?

Consider the n-dimensional ‘table’ for a constraint on n variables
\( v_1, v_2, \ldots, v_n \).

To check all domain elements for support we may need to traverse the whole space
\[ D(v_1) \times D(v_2) \times \cdots \times D(v_n). \]

Hence, in general it is intractable to make non-binary constraints domain consistent!

Notorious Example: the constraint \[ \Sigma_{i=1..n} c_i \cdot x_i = b \]
(c_i and b are integer constants, x_i integer variables (i=1..n)).
Fortunately, some non-binary constraints can be made domain consistent in polynomial time!

**Example:** *alldifferent*(\(x_1, x_2, \ldots, x_n\))

A solution to this constraint has a special structure\(^1\), and there exists an efficient algorithm to find it.

\(^1\) See next lecture.
Bounds consistency

In many cases we do not know a special structure to exploit. We can sometimes handle those difficult non-binary constraints by relaxing the consistency.

Instead of domain consistency, we demand for example bounds consistency.

**Definition:** a constraint $C$ is *bounds consistent* if the bounds of all variable domains belong to a solution to $C$, where all domains are treated as intervals $[\min(D(x)), \max(D(x))]$.

(Note: we assume that the domains are all drawn from a totally ordered ground set.)

**Example:** $\Sigma_{i=1..n} c_i \cdot x_i = b$

can be made bounds consistent in polynomial time.
**Theorem:** If all constraints can be made consistent in poly-time, then the Propagation Cycle runs in poly-time.

**Proof:**

```c
bool Propagation_Cycle(V, D, C) {
    Q := V;
    while Q nonempty {
        pick v in V;
        Q := Q - v;
        for all constraints C in C containing v {
            if (Propagate(C, U, D) = false) return false;
            if a domain of variable x has changed, set Q := Q + x;
        }
    }
    return true;
}
```

In each loop, either $|Q| := |Q| - 1$, or some domain element is removed. So there are at most $n + n \cdot d$ loops (for $n$ variables with maximum domain size $d$).
Recall that we can model the graph coloring problem as

variables $\mathcal{V}$: $x_1, x_2, ..., x_8$

domains $\mathcal{D}$: \{r,b,g,p\}

constraints $\mathcal{C}$: $\text{alldifferent}(\mathcal{V}_K)$ for all cliques $K$ in the graph.

Domain consistency for $\text{alldifferent}$ is poly-time,
Propagation Cycle is poly-time,
$\rightarrow$ graph coloring is poly-time?

But as graph coloring is NP-hard,
do we have $P = NP$?

No, finding maximum clique is NP-hard!
Moreover, even if we have *alldifferent* constraints for all cliques, it is not sufficient to determine consistency of the CSP:

![Diagram](image-url)
Moreover, even if we have \textit{alldifferent} constraints for all cliques, it is not sufficient to determine consistency of the CSP:
Moreover, even if we have *alldifferent* constraints for *all* cliques, it is not sufficient to determine consistency of the CSP:
Moreover, even if we have *alldifferent* constraints for *all* cliques, it is not sufficient to determine consistency of the CSP:

But there is no solution!
Moreover, even if we have *alldifferent* constraints for *all* cliques, it is not sufficient to determine consistency of the CSP:

```
try

but there is no solution!
```
Moreover, even if we have *alldifferent* constraints for all cliques, it is not sufficient to determine consistency of the CSP:

but there is no solution!
Summary of Complexity of CSP Solving


However, with constraint programming we can control (to some extent) where to put the NP-hardness:

- exponential-size search tree,
- exponential-time (but maybe effective) propagation algorithm,
- exponential-time to create the model.

Moreover, we have a tool to exploit as much tractability as possible:

- e.g., collect information of tractable substructures in global constraints and perform poly-time constraint propagation.

Still, even if all propagation is tractable, in general the search tree has exponential size in the worst case.

(See also lectures by Jeavons on Constraints and Complexity)
Additional resources

“Handbook of Constraint Programming,” edited by F. Rossi, P. van Beek, T. Walsh

“Constraint-Based Local Search”, by Pascal Van Hentenryck and Laurent Michel

Integrated Methods for Optimization, by John N. Hooker

“Programming with Constraints,” by Kim Marriott, Peter J. Stuckey

“Principles of Constraint Programming,” by Krzysztof Apt

“Constraint Processing,” by Rina Dechter