Constraint Satisfaction and Constraint Programming

Pedro Meseguer
IIIA-CSIC
Bellaterra, Spain

Overview

Introduction

Constraint Satisfaction
• Search
• Inference
• Hybrids

Constraint Programming
• Modelling with Constraints
• Optimization
• Existing Solvers

Summary
Modern Art

How can we reconstruct the painting?

Modern Art: Accident
Modern Art: Reconstruction

1. Locate pieces in grid slots
2. Two adjacent slots must have the same color pattern on the contact edge
3. Find a globally consistent arrangement

Modern Art: All Constraints

12 constraints

Solution: assignment satisfying every constraint
Modern Art: Solution

Conclusions from Modern Art

Constraint problems: most of the knowledge can be expressed in terms of constraints among problem elements

One constraint:
- Involves a subset of problem elements
- Declares permitted (or forbidden) value combinations
- Provides a local view of the whole problem

Solution:
- Satisfies every constraint
- Global view of the whole problem
- Process: from local to global consistency
Overview

Introduction
Constraint Satisfaction
  • Search
  • Inference
  • Hybrids
Constraint Programming
  • Modelling with Constraints
  • Optimization
  • Existing Solvers
Summary

Some Definitions

Constraint Network (CN): \((X, D, C)\)
  • \(X = \{x_1, x_2, \ldots, x_n\}\) variables
  • \(D = \{d_1, d_2, \ldots, d_n\}\) domains (finite)
  • \(C = \{c_1, c_2, \ldots, c_r\}\) constraints
    \(c \in C\)
    \(\text{var}(c) = \{x_i, x_j, \ldots, x_k\}\) scope
    \(\text{rel}(c) \subseteq d_i \times d_j \times \ldots \times d_k\) permitted tuples

Constraint Satisfaction Problem (CSP):
  • CN solving: assignment satisfying every constraint
  • NP-complete task
Relevance

CSP: formal model to express problems

Many problems can be represented as CSP:
- Academic problems:
  - SAT, Graph coloring, N-queens, . . .
- Real problems:
  - Scheduling, Resource allocation, Routing, ....

Many AI tasks can be modeled as CSP:
- Automated reasoning
- Planning
- Spatial and temporal inference

Running Example: n-queens

GOAL: Locate n queens in an n x n chessboard, such that they do not attack each other

Formulation:
- Variables: one queen per row
- Domains: available columns
- Constraints:
  - different columns and different diagonals
  \[ x_i \neq x_j \quad \land \quad |x_i - x_j| \neq |i - j| \]

Constraint Graph:
Backtrack Search

Strategy:
• Build a partial solution:
  • A partial consistent assignment
• Extend consistently the partial solution
  • One new assigned variable each time
• If no consistent extension:
  • Backtrack: change a previous assignment

Variables:
• Past ∈ partial solution (assigned)
• Future ∉ partial solution (unassigned)

Tree Search

State space: explored as a tree
• root: empty
• one variable per level
• successors of a node:
  • one successor per value of the variable
  • meaning: variable ← value

Tree:
• each branch defines an assignment
• depth n (number of variables)
• branching factor d (domain size)
Search tree for 4-queens

Backtracking Algorithm

Depth-first tree traversal (DFS)

At each node:
- check every completely assigned constraint
- if consistent, continue DFS
- otherwise, prune current branch
- continue DFS

Complexity: $O(d^n)$
Backtracking on 4-queens

- $x_1$
- $x_2$
- $x_3$
- $x_4$

25 nodes

Solution

Problems of Backtracking

Thrashing:
- the same failure can be rediscovered an exponential number of times

Solutions:
- check not completely assigned constraints: lookahead
- non-chronological backtracking: backjumping

The first choice is incompatible with any last choice
**Backjumping**

Non-chronological backtracking:
- jumps to the last decision responsible for the dead-end
- intermediate decisions are removed

```
\{x_1 \leftarrow 1\}
\{x_3 \leftarrow 2\}
\{x_2 \leftarrow 4\}
```

```
\{x_1 \leftarrow 1\}
\{x_2 \leftarrow 4\}
\{x_3 \leftarrow 2\}
```

**Inference**

Inference: \( P \rightarrow P' \)

- \( P' \) is equivalent to \( P \): \( \text{Sol}(P) = \text{Sol}(P') \)
- \( P' \) is presumably easier to solve than \( P \)
  - smaller search space
  - constraints are more explicit

Inference can be:
- complete: produces the solution
  adaptive consistency
- incomplete: requires further search
  arc consistency
Adaptive Consistency

Problem $P$, var $x$, $C_x=${constraints on $x$}

Idea:
- Substitute $C_x$ by a new constraint $c$
- $c$ summarizes the effect of $C_x$ on $P$
- $c$ does not mention $x$

now $x$ is isolated: it can be eliminated

Process:
- problems: $P \rightarrow P' \rightarrow P'' \rightarrow \ldots \rightarrow P^{(n-1)}$
- #vars: $n \rightarrow n-1 \rightarrow n-2 \ldots \rightarrow 1$

solution without search

Variable Elimination

To eliminate var $x$:
- Join all constraints in $C_x \rightarrow c$
- Substitute $C_x$ by $c$
- Project out variable $x$ from $c \rightarrow c$
- Substitute $c$ by $c$

Example: 4-queens ($x_1$)

\begin{center}
\begin{tabular}{|c|c|c|}
\hline
$x_2$ & $x_3$ & $x_4$ \\
\hline
3 & 2 & 2 \\
3 & 4 & 2 \\
3 & 4 & 3 \\
4 & 2 & 2 \\
4 & 2 & 3 \\
4 & 4 & 2 \\
4 & 4 & 3 \\
4 & 1 & 1 \\
4 & 1 & 3 \\
4 & 1 & 4 \\
4 & 3 & 1 \\
4 & 3 & 3 \\
4 & 3 & 4 \\
\hline
\end{tabular}
\end{center}

Example: 4-queens ($x_2$)

\begin{center}
\begin{tabular}{|c|c|}
\hline
$x_3$ & $x_4$ \\
\hline
4 & 2 \\
4 & 2 \\
4 & 3 \\
4 & 1 \\
\hline
\end{tabular}
\end{center}
Example: 4-queens ($x_3$)

Example: All Solutions 4-queens

SOLUTION 1: 2 4 1 3
**Example: All Solutions 4-queens**

SOLUTION 2:

```
  x4  x3  x2  x1
  3   1   4   2
```

**Arc Consistency**

- c is arc-consistent iff: every possible value of every variable in var (c) appears in rel(c)

- If c is not arc-consistent because a ∈ d_x:
  - a will not be in any solution
  - a can be removed: d_x ← d_x – {a}
  - if d_x becomes empty, P has no solution

- P is arc-consistent iff: every constraint is arc-consistent

- If P is arc-consistent → P has solution
Example: 3-queens

\[ c_{12} \text{ is not arc-consistent because value 2 of } d_1 \]

\[ c_{12} \text{ is not arc-consistent because value 2 of } d_2 \]

\[ c_{23} \text{ is not arc-consistent because value 2 of } d_3 \]

Constraint Propagation

- \( AC(c) \): procedure to make \( c \) arc consistent
- To make \( P \) arc-consistent, process each constraint
- But \( AC(c) \) may render other constraints arc-inconsistent
- To make \( P \) arc-consistent, iterate:
  - Apply \( AC \) on \( \{c_1, c_2, \ldots, c_r\} \)
  - Until no changes in domains: fix point
Example: 3-queens

value 2 of $d_4$ was removed
(to make $c_{23}$ arc-consistent)

this makes $c_{13}$ arc-inconsistent

c_{13} is not arc-consistent
because value 1 of $d_1$

c_{13} is not arc-consistent
because value 3 of $d_1$

domain $d_1$ empty

no solution !!

Hybrids: Search + Inference

Idea:
• Search: backtracking (could be non-chronological)
• Inference: at each node, AC on some constraints
  • Future domains are pruned
  • Values no AC are eliminated

Effect:
• Future domains are reduced: less nodes to explore
• AC at each node: more work per node
• Very beneficial: reduces thrashing
Forward Checking

FC is a combination of:
- Search: backtracking
- Inference: at each node, AC on constraints with assigned and unassigned variables

When a domain becomes empty:
- No solutions following current branch
- Prune current branch and backtrack

Caution:
- Values removed by AC at level i, have to be restored when backtracking at level i or above

Example: FC on 4-queens

8 nodes

solution
Maintaining Arc Consistency

MAC is a combination of:
• Search: backtracking
• Inference: at each node, AC on all constraints
• Preprocess: subproblems are AC

When a domain becomes empty:
• No solutions following current branch
• Prune current branch and backtrack

Caution:
• Values removed by AC at level i, have to be restored when backtracking at level i or above

Example: MAC on 4-queens

5 nodes
solution
Search Heuristics

Dynamic variable selection:
- Variable may have different orders in branches
- Freedom to choose next variable

Heuristic:
1. Select the variable with minimum domain
2. Select the variable involved in most constraints

Combination: \( \min \left( \frac{\text{domain}}{\text{degree}} \right) \)

Overview

Introduction
Constraint Satisfaction
- Search
- Inference
- Hybrids

Constraint Programming
- Modelling with Constraints
- Optimization
- Existing Solvers

Summary
Modelling

Problem P as CSP:
• Several formulations are possible
• Select variables and domains
  • Search space size: $|d_1| \times |d_2| \times \ldots \times |d_n|$
  • Select formulation with smallest size
• Select constraints:
  • Number of constraints
  • Arity
  • AC cost
  • Pruning power

Constraints

Number:
• High: causes a high overhead
• Low: is preferred (compact representation)
• Keeping low number, some redundancy is advised

Arity: number of variables involved in a constraint

Arity and AC:
• $\text{arity}(c) = k$, $\text{AC}(c)$ is $O(d^k)$
• high arity causes higher AC cost
• but AC on high arity constraints prunes more !!
Global Constraints

c is global iff:
- arity(c) > 2
- c is logically equivalent to \{c_1, c_2, \ldots, c_k\} binary
- AC(c) prunes more than AC(c_1, c_2, \ldots, c_k)

Propagation:
- specialized algorithms decrease AC complexity
- exploits the constraint semantics

Catalog:
- set of common structures to reuse
- best known algorithms for propagation

Example: all-different

3 binary constraints, they are AC, no pruning

1 ternary constraint, it is not AC, AC pruning → empty domain, no solution!!
Optimization

Constraint Optimization Problem: \((X, D, C, F)\)
\[ F(X) \text{ is a cost function} \]

GOAL: \( \min F(X), \) satisfying \( C \)

Solving method:
- Hybrid: search + consistency assigned constraints
  - DFS
- When \( F(X) = Z^* \), add constraint \( F(X) < Z^* \)

Branch and Bound

Search: depth-first

At each node:
- Consistency on assigned constraints
- AC on (some) constraints (optional)
- Computes a lower bound of \( F(X) \): \( F(X) \)

Prunes current branch: when
- Inconsistent assigned constraint
- Empty domain (because AC)
- \( F(X) > Z^* \): no solution will improve \( Z^* \)
Constraint Programming

Declarative Programming: you declare
- Variables
- Domains
- Constraints
and ask the SOLVER to find a solution!!

SOLVER offers:
- Implementation for variables / domains / constraints
- Hybrid algorithm: backtracking + incomplete inference
- Global constraints + optimized AC propagation
- Empty domain detection
- Embedded heuristics

Constraint Logic Programming

Logic Programming:
- Depth-first search
- Unification: substitute equals by equals clauses/database

Existing solvers:
- Chip, Eclipse, Mozart, Sictus Prolog (and many others)
Imperative Constraint Programming

Library to be included in your program

Provides:
• Special objects:
  • Variables / Domains / Constraints (global)
• Special functions to find:
  • One solution / the next solution

Existing Solvers:
• Ilog Solver, Choco

Summary

Constraint Satisfaction
• Search: backtracking
• Inference: complete / incomplete (AC)
• Hybrids: backtracking + AC

Constraint Programming
• Modelling: formulation / global constraints
• Optimization: branch and bound
• Existing Solvers: logical vs imperative CP
To know more . . .

Next week, slides and a list of references available at

http://www.iiia.csic.es/~pedro/