Knowledge Representation and Reasoning

Constraint Reasoning

Prof. Dr. Hans Jürgen Ohlbach
Contents

This lecture
• Motivation
• Constraint Types
• Constraint Satisfaction Problems
• Constraint Propagation
• Arc-Consistency, Path-Consistency
• Choice Constraint-Propagation Backtracking
• Constraint Optimization

Further Lectures
• Allen's Interval Calculus for temporal intervals
• Constraint Handling Rules
**Starting Point**

non-deterministic algorithms for search problems:
- guess a potential solution
- check if the guess is okay

Typical non-deterministic approach for NP-complete problems.

**Example:** 3-colorability of a nodes of a graph, adjacent nodes must not have the same color
How Does one Guess?

**Choice and Backtracking:**

- make a (blind) choice
- check whether the choice is okay
  If not, backtrack to the last choice-point and choose an alternative
- make another choice, until a solution is found, or there are no alternatives any more
The Graph Coloring Example

illegal: backtracking starts now
Choice, Constraint Propagation and Backtracking:

- make a (blind) choice
- Compute the consequences of the choice (constraint propagation, CP)
- If the consequences lead to a contradiction, backtrack to the last choice point
The Graph Coloring Example with CP

No backtracking in this simple example
N-Queens Problem
Place n queens on a chess board such that no queen threatens another one.

Choice-CP-Backtrack solution:
• place a queen
• mark all rows, columns, diagonals, threatened by the queen as forbidden
• place another queen at a free place
• If there is no free place, backtrack

http://www.logic.at/prolog/queens/queens.html
Other Search-CP-Backtracking Problems

- Crossword Puzzles
- Sudoku
- Time Tabelling
  Generate a time table for a school, university, ...
- Configuration and Design
- Scheduling Problems
- SAT-Solving
- Any NP-complete problem
CP-Systems

- Constraint-Processing integrated in concrete algorithms (e.g. special SAT-solver)
- Constraint Networks, i.e. Constraint-Processing libraries for classes of constraint-systems (finite domain constraint logic programming clp(FD) for Prolog)
- Constraint-Processing and backtracking integrated (e.g. in Prolog)
- Constraint Optimization (for hard- and soft-constraints)
- Constraint Handling Rules (CHR) as a system for programming constraint handlers
- CHR integrated in a host language (for backtracking)
A constraint network \((X,D,C)\) consists of

- A finite set of variables \(X_1,...,X_n\)
- For each variable \(X_i\) an associated domain \(D_i\)
- A set of relations (constraints) between the variables.

**Example:**
a relation \(R_{X_1X_3} = \{(1,3),(2,5)\}\) means that \(X_1 = 1\) and \(X_3 = 3\), as well as \(X_1 = 2\) and \(X_3 = 5\) are legal combinations.

\(X_1X_3\) is in this case the **scope** of the relation \(R_{X_1X_3}\)

If all constraint are **binary relations** then the constraint network is a **Binary Constraint Network**
Example: n-Queens Problem

We exploit that each row must contain exactly 1 queen

**Variables:** $Q_1, \ldots, Q_n$ ($Q_i$ denotes the queen in row $i$)

**Domains:** each queen can be in column $1, \ldots, n$, therefore $D_i = \{1, \ldots, n\}$

$Q_5 = 6$ means the queen at row 5 is in column 6.

**Constraints:** a binary constraint $R_{ij} = \{(\ldots,\ldots),\ldots\}$ expresses the legal positions of queen $i$ relative to queen $j$.

Example:

$R_{12} = \{(1,3),(1,4),\ldots,(1,n),(2,4),\ldots,(2,n)\ldots(n-2,n),(n,n-2)\ldots,(3,1)\}$

Similar constraints must be defined for all pairs of queens.
A binary constraint network can be represented as a graph (the constraint graph).

- The nodes are the variables.
- Two nodes are connected by an arc if the variables are related by a constraint.

In the graph coloring problem the graph itself is the constraint graph.
Constraint Graph for the 4-queens problem

The constraints are:

R12 = {(1,3)(1,4)(2,4)(4,2)(4,1)(3,1)}
R13 = {(1,2)(1,4)(2,1)(2,3)(3,2)(3,4)(4,1)(4,3)}
R14 = {(1,2)(1,3)(2,1)(2,3)(2,4)(3,1)(3,2)(3,4)(4,2)(4,3)}
R23 = {(1,2)(1,4)(2,4)(3,1)(4,1)(4,2)}
R24 = {(1,2)(1,4)(2,1)(2,3)(3,2)(3,4)(4,1)(4,3)}
R34 = {(1,3)(1,4)(2,4)(3,1)(4,1)(4,2)}
Instantiation

An **instantiation** of a variables is an assignment of their domain values to the variables. \( Q_3 = 2, Q_4=1 \) in the 4-queens problem, for example is an instantiation.

An instantiation **satisfies the constrains**, if the corresponding tuples are in the corresponding relations.

**Example** (4-queens problem):

\( Q_1 = 1, Q_2 = 4 \) satisfies the constrains because \((1,4) \in R_{12}\)

\( Q_1 = 1, Q_2 = 2 \) does not satisfy the constrains because \((1,2) \notin R_{12}\)
An **solution** of a Constraint Satisfaction Problem \((V,D,C)\) is an instantiation of **all** variables that satisfies **all** constraints.

**Example** (4-queens problem):

\(Q_1 = 1, \ Q_2 = 3, \ Q_3 = 2, \ Q_4 = 4\) is a solution

\(Q_1 = 3, \ Q_2 = 1, \ Q_3 = 4, \ Q_4 = 2\) is another solution
Given a Constraint Satisfaction Problem (V,D,C), where, for example, the variable X has domain \{1,\ldots,8\}, the \textit{reduction} of the possible values for X, say X \in \{2,\ldots,8\} may cause that certain constraints are no longer satisfied, and therefore a related variable Y may get its possible values restricted.

Computing these restrictions is called \textit{Constraint Propagation} (CP)

\textbf{Example:} 3-coloring of graphs

![Diagram](image_url)
Arc Consistency

The simplest and most efficient way of Constraint Propagation is to propagate the constraints \textit{just to the neighbour variables} in the constraint graph. If this is done exhaustively, the final graph is called \textit{arc-consistent}.

Example: this graph for the 3-coloring problem is arc consistent. There is nothing to propagate further.
An arc-consistent graph may in fact have no solution.

**Example:** this graph for the 2-coloring problem is arc consistent. There is nothing to propagate further, but the graph cannot be 2-colored.
Path Consistency

Path-consistency propagates constraints over two arcs.

Example: Consider again the 2-coloring problem:

The constraints are: \( R_{12} = R_{13} = R_{23} = \{ (g,b),(b,g) \} \)
The composition \( R_{12} \circ R_{23} = \{ (g,g),(b,b) \} \)
However, \( (R_{12} \circ R_{23}) \cap R_{23} = \emptyset. \)
This means, the graph cannot be 2-colored.
Method:
As long as there are no changes any more
- choose constraints $R_{XY}$ and $R_{YZ}$
- set the new value for $R_{XZ}$ to $(R_{XY} \circ R_{YZ}) \cap R_{XZ}$.
- If the new $R_{XZ}$ became empty, stop with 'no solution'.

If the method did not terminate with 'no solution' then the final network is called path consistent.

Good implementations achieve a complexity of $O(n^3k^5)$ where $n$ is the number of variables and $k$ is the maximum domain size.
The following example shows that a path consistent network need not have a solution

**Example**: 3-coloring of a graph
This graph is path-consistent, but cannot be 3-colored.
Consequence

In order to get completeness, i.e. to guarantee that an inconsistency (no solution possible) is detected, one must combine constraint propagation by arc or path consistency with a choice and backtracking procedure.

It is always a challenge to find problem classes where arc or path consistency is complete (for NP-complete problems only if P = NP).
Global Constraints

In principle, all n-ary relations can be mapped to binary relations. But this may become very inefficient. Therefore most constraint handling systems allow for so called **global constraints**.

A global constraint involves a larger set, or even all variables.

A typical example is \( \text{all-different}(X_1, \ldots, X_n) \)

enforcing the assignment to \( X_1, \ldots, X_n \) to be different from each other.

This is, for example, very useful for the n-queens problem (all queens must be in a different column).

In the path consistency algorithm, this can be incorporated by automatically removing all relations \( R_{XY}(x,x) \)
Implementation

An extremely good implementation of constraint propagation within Prolog is the clp(fd) library for *finite domains* (the values of the variables are finite sets of numbers)

See, for example
https://sicstus.sics.se/sicstus/docs/3.7.1/html/sicstus_33.html#SEC265
or
http://www.swi-prolog.org/man/clpfd.html

The implementation of the n-queens problem
http://www.logic.at/prolog/queens/queens.html
shows the power of the system.
100 queens are no problem.
Many problems have constraints which need not necessarily be satisfied.

**Example:** Timetabling

**Hard-constraints:** A lecturer cannot give two lectures at the same time

**Soft-constraints:** The lectures of Prof. Smith should be in the morning.

A solution to the constraint problem should satisfy all hard constraints, and the soft constraints as good as possible (a lecture of Prof. Smith 11-13 o'clock might still be acceptable).
Formalizing Soft Constraints

Soft-Constraints can be formalized by specifying functions $F(x_1,\ldots,x_k)$ and minimizing/maximizing these functions.

**Example**: timetabelling (Prof. Smith wants his lectures in the morning)
If $X_1, X_2, X_3$ represent Prof. Smith's lectures then $F_{\text{Smith}}(X_1, X_2, X_3)$ could be

$$F_{\text{Smith}}(X_1, X_2, X_3) = \sum_{i=0}^{3} \max(0, \text{end-time}(X_i) - \text{noon})$$

which has to minimized.

In general, one has several of these functions, and tries to minimize/maximize the sum of all these functions (this is the **cost function**).
Constraint Optimization as a sequence of Constraint Satisfaction Problems.

1) Solve the constraint satisfaction part (CSP) without considering the cost function
2) Compute the cost function $F$, the value is, say $C$
3) Add a new hard-constraint $F > C$
4) Solve the CSP with the new hard-constraint. If there is no solution, then the previous solution was optimal, stop
5) Compute the cost function again, say $F = D$
6) Change the hard constraint to $F > D$ and go to 4).

Various optimizations of this simple method have been developed.
Summary

We have introduced

- Constraint Satisfaction Problems
- Constraint Propagation
- Arc-Consistency
- Path-Consistency
- Incompleteness of these methods
- Combination: Choice Constraint-Propagation Backtracking
- Constraint Optimization Problems