Artificial Intelligence

Constraint Satisfaction Problems

Marc Toussaint
University of Stuttgart
Winter 2015/16

(slides based on Stuart Russell’s AI course)
Inference

- The core topic of the following lectures is
  **Inference**: Given some pieces of information on some things (observed variables, prior, knowledge base) what is the implication (the implied information, the posterior) on other things (non-observed variables, sentence)

- Decision-Making and Learning can be viewed as Inference:
  - given pieces of information: about the world/game, collected data, assumed model class, *prior* over model parameters
  - make decisions about actions, classifier, model parameters, etc

- In this lecture:
  - “Deterministic” inference in CSPs
  - Probabilistic inference in graphical models variables
  - Logic inference in propositional & FO logic
Constraint satisfaction problems (CSPs)

- In previous lectures we considered sequential decision problems. CSPs are not sequential decision problems. However, the basic methods address them by testing sequentially 'decisions'.

CSP:
- We have $n$ variables $x_i$, each with domain $D_i$, $x_i \in D_i$
- We have $K$ constraints $C_k$, each of which determines the feasible configurations of a subset of variables
- The goal is to find a configuration $X = (X_1, \ldots, X_n)$ of all variables that satisfies all constraints

Formally $C_k = (I_k, c_k)$ where $I_k \subseteq \{1, \ldots, n\}$ determines the subset of variables, and $c_k : D_{I_k} \rightarrow \{0, 1\}$ determines whether a configuration $x_{I_k} \in D_{I_k}$ of this subset of variables is feasible.
Example: Map-Coloring

Variables $W, N, Q, E, V, S, T$  \hspace{1cm} (\textit{E} = \text{New South Wales})

Domains $D_i = \{\text{red, green, blue}\}$ for all variables

Constraints: adjacent regions must have different colors

\hspace{1cm} e.g., $W \neq N$, or

$(W, N) \in \{(\text{red, green}), (\text{red, blue}), (\text{green, red}), (\text{green, blue}), \ldots\}$
Example: Map-Coloring contd.

Solutions are assignments satisfying all constraints, e.g.,
\[ \{W = \text{red}, N = \text{green}, Q = \text{red}, E = \text{green}, V = \text{red}, S = \text{blue}, T = \text{green}\} \]
Constraint graph

- **Pair-wise CSP**: each constraint relates at most two variables
- **Constraint graph**: a *bi-partite graph*: nodes are variables, boxes are constraints
- In general, constraints may constrain several (or one) variables \(|I_k| \neq 2\)

![Constraint graph diagram]
Varieties of CSPs

- **Discrete variables**: finite domains; each $D_i$ of size $|D_i| = d \Rightarrow O(d^n)$ complete assignments
  - e.g., Boolean CSPs, incl. Boolean satisfiability
  - infinite domains (integers, strings, etc.)
  - e.g., job scheduling, variables are start/end days for each job
  - linear constraints solvable, nonlinear undecidable

- **Continuous variables**
  - e.g., start/end times for Hubble Telescope observations
  - linear constraints solvable in poly time by LP methods

- **Real-world examples**
  - Assignment problems, e.g. who teaches what class?
  - Timetabling problems, e.g. which class is offered when and where?
  - Hardware configuration
  - Transportation/Factory scheduling
Varieties of constraints

**Unary** constraints involve a single variable, $|I_k| = 1$
  e.g., $S \neq \text{green}$

**Pair-wise** constraints involve pairs of variables, $|I_k| = 2$
  e.g., $S \neq W$

**Higher-order** constraints involve 3 or more variables, $|I_k| > 2$
  e.g., Sudoku
Methods for solving CSPs
Sequential assignment approach

Let’s start with the straightforward, dumb approach, then fix it
States are defined by the values assigned so far

- **Initial state**: the empty assignment, \( \{ \} \)
- **Successor function**: assign a value to an unassigned variable that does not conflict with current assignment \( \Rightarrow \) fail if no feasible assignments (not fixable!)
- **Goal test**: the current assignment is complete

1) Every solution appears at depth \( n \) with \( n \) variables \( \Rightarrow \) use depth-first search
2) \( b = (n - \ell)d \) at depth \( \ell \), hence \( n!d^n \) leaves!
Backtracking sequential assignment

- Two variable assignment decisions are **commutative**, i.e.,
  \[
  [W = \text{red} \text{ then } N = \text{green}] \text{ same as } [N = \text{green} \text{ then } W = \text{red}]
  \]
- We can fix a single next variable to assign a value to at each node
- This does not compromise completeness (ability to find the solution)
  \[\Rightarrow b = d \text{ and there are } d^n \text{ leaves}\]

- Depth-first search for CSPs with single-variable assignments is called **backtracking** search
- Backtracking search is the basic uninformed algorithm for CSPs

Can solve $n$-queens for $n \approx 25$
Backtracking search

function Backtracking-Search(csp) returns solution/failure
    return Recursive-Backtracking({}, csp)

function Recursive-Backtracking(assignment, csp) returns soln/failure
    if assignment is complete then return assignment
    var ← Select-Unassigned-Variable(VARIABLES[csp], assignment, csp)
    for each value in Ordered-Domain-Values(var, assignment, csp) do
        if value is consistent with assignment given CONSTRAINTS[csp] then
            add [var = value] to assignment
            result ← Recursive-Backtracking(assignment, csp)
            if result ≠ failure then return result
            remove [var = value] from assignment
    return failure
Backtracking example
Backtracking example
Backtracking example
Backtracking example
Improving backtracking efficiency

Simple heuristics can give huge gains in speed:

1. Which variable should be assigned next?
2. In what order should its values be tried?
3. Can we detect inevitable failure early?
4. Can we take advantage of problem structure?
Minimum remaining values

Minimum remaining values (MRV):
choose the variable with the fewest legal values
Degree heuristic

Tie-breaker among MRV variables

Degree heuristic:
choose the variable with the most constraints on remaining variables
Least constraining value

Given a variable, choose the least constraining value: the one that rules out the fewest values in the remaining variables.

Combining these heuristics makes 1000 queens feasible.
Constraint propagation

- After each decision (assigning a value to one variable) we can compute what are the remaining feasible values for all other variables.
- Initially, every variable has the full domain $D_i$. Constraint propagation reduces these domains, deleting entries that are inconsistent with the new decision. These dependencies are recursive: Deleting a value from the domain of one variable might imply infeasibility of some value of another variable → constraint propagation. We update domains until they’re all consistent with the constraints.

*This is Inference*
Constraint propagation

- Example of just “1-step propagation”:

*N* and *S* cannot both be blue!

Idea: propagate the implied constraints several steps further

Generally, this is called **constraint propagation**
Arc consistency (=constraint propagation for pair-wise constraints)

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

for every value $x$ of $X$ there is some allowed $y$
Arc consistency (=constraint propagation for pair-wise constraints)

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

for every value $x$ of $X$ there is some allowed $y$
Arc consistency (=constraint propagation for pair-wise constraints)

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

for every value $x$ of $X$ there is some allowed $y$

If $X$ loses a value, neighbors of $X$ need to be rechecked
Arc consistency (=constraint propagation for pair-wise constraints)

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

for every value $x$ of $X$ there is some allowed $y$

If $X$ loses a value, neighbors of $X$ need to be rechecked

Arc consistency detects failure earlier than forward checking

Can be run as a preprocessor or after each assignment
Arc consistency algorithm (for pair-wise constraints)

function AC-3( csp) returns the CSP, possibly with reduced domains
inputs: csp, a pair-wise CSP with variables \{X_1, X_2, \ldots, X_n\}
local variables: queue, a queue of arcs, initially all the arcs in csp

while queue is not empty do
    (X_i, X_j) ← REMOVE-FIRST(queue)
    if REMOVE-INCONSISTENT-VALUES(X_i, X_j) then
        for each X_k in NEIGHBORS[X_i] do
            add (X_k, X_i) to queue

function REMOVE-INCONSISTENT-VALUES( X_i, X_j) returns true iff Dom[X_i] changed
changed ← false
for each x in Domain[X_i] do
    if no value y in Domain[X_j] allows (x,y) to satisfy the constraint X_i ↔ X_j
        then delete x from Domain[X_i]; changed ← true
return changed

\(O(n^2d^3)\), can be reduced to \(O(n^2d^2)\)
Constraint propagation

See textbook for details for non-pair-wise constraints
Very closely related to message passing in probabilistic models

In practice: design approximate constraint propagation for specific problem
  E.g.: Sudoku: If $X_i$ is assigned, delete this value from all peers
Problem structure

Tasmania and mainland are independent subproblems
Identifiable as connected components of constraint graph
Tree-structured CSPs

Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n d^2)$ time

Compare to general CSPs, where worst-case time is $O(d^n)$

This property also applies to logical and probabilistic reasoning!
Algorithm for tree-structured CSPs

1. Choose a variable as root, order variables from root to leaves such that every node’s parent precedes it in the ordering

![Tree diagram]

2. For \( j \) from \( n \) down to 2, apply

\[
\text{RemoveInconsistent}(\text{Parent}(X_j), X_j)
\]

This is *backward constraint propagation*

3. For \( j \) from 1 to \( n \), assign \( X_j \) consistently with \( \text{Parent}(X_j) \)

This is *forward sequential assignment* (trivial backtracking)
Nearly tree-structured CSPs

**Conditioning:** instantiate a variable, prune its neighbors’ domains

---

**Cutset conditioning:** instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree

Cutset size $c \Rightarrow$ runtime $O(d^c \cdot (n - c)d^2)$, very fast for small $c$
Summary

• CSPs are a fundamental kind of problem: finding a feasible configuration of \(n\) variables the set of constraints defines the (graph) structure of the problem

• Sequential assignment approach

  Backtracking = depth-first search with one variable assigned per node

• Variable ordering and value selection heuristics help significantly

• Constraint propagation (e.g., arc consistency) does additional work to constrain values and detect inconsistencies

• The CSP representation allows analysis of problem structure

• Tree-structured CSPs can be solved in linear time

  If after assigning some variables, the remaining structure is a tree
  \(\rightarrow\) linear time feasibility check by tree CSP