Foundations of Artificial Intelligence

5. Constraint Satisfaction Problems

CSPs as Search Problems, Solving CSPs, Problem Structure

Joschka Boedecker and Wolfram Burgard and Bernhard Nebel

Albert-Ludwigs-Universität Freiburg

May 12, 2017
Contents

1. What are CSPs?
2. Backtracking Search for CSPs
3. CSP Heuristics
4. Constraint Propagation
5. Problem Structure
A Constraint Satisfaction Problems (CSP) is given by

- a set of **variables** \( \{x_1, x_2, \ldots, x_n\} \),
- an associated set of **value domains** \( \{\text{dom}_1, \text{dom}_2, \ldots, \text{dom}_n\} \), and
- a set of **constraints**. i.e., relations, over the variables.

An **assignment** of values to variables that **satisfies** all constraints is a solution of such a CSP.

In CSPs viewed as search problems, states are explicitly represented as variable assignments. CSP search algorithms take advantage of this structure.

The main idea is to exploit the constraints to eliminate large portions of search space.

**Formal representation language** with associated general inference algorithms.
Example: Map-Coloring

- **Variables:** WA, NT, SA, Q, NSW, V, T
- **Values:** \{red, green, blue\}
- **Constraints:** adjacent regions must have different colors, e.g., NSW \neq V
Australian Capital Territory (ACT) and Canberra (inside NSW)

View of the Australian National University and Telstra Tower
Solution assignment:

\[ \{\text{WA} = \text{red}, \text{NT} = \text{green}, \text{Q} = \text{red}, \text{NSW} = \text{green}, \text{V} = \text{red}, \text{SA} = \text{blue}, \text{T} = \text{green}\} \]

Perhaps in addition \( \text{ACT} = \text{blue} \)
a constraint graph can be used to visualize binary constraints
for higher order constraints, hyper-graph representations might be used
Nodes = variables, arcs = constraints

Note: Our problem is three-colorability for a planar graph
Variations

- Binary, ternary, or even higher **arity** (e.g., ALL_DIFFERENT)

- **Finite** domains ($d$ values) $\rightarrow d^n$ possible variable assignments

- **Infinite** domains (reals, integers)
  - *linear constraints (each variable occurs only in linear form)*: solvable (in P if real)
  - *nonlinear constraints*: unsolvable
Applications

- Timetabling (classes, rooms, times)
- Configuration (hardware, cars, . . . )
- Spreadsheets
- Scheduling
- Floor planning
- Frequency assignments
- Sudoku
- . . .
Backtracking Search over Assignments

- Assign values to variables step by step (order does not matter)
- Consider only one variable per search node!
- **DFS** with single-variable assignments is called **backtracking search**
- Can solve $n$-queens for $n \approx 25$
Algorithm

\begin{algorithm}
\begin{algorithmic}
\Function{BACKTRACKING-SEARCH}{csp} \Return a solution, or failure 
\State \textbf{return} BACKTRACK(\{\}, csp)
\end{algorithmic}
\end{algorithm}

\begin{algorithmic}
\Function{BACKTRACK}{assignment, csp} \Return a solution, or failure 
\If{assignment is complete} \textbf{return} assignment 
\State \textit{var} $\leftarrow$ \Call{SELECT-UNASSIGNED-VARIABLE}{csp} 
\For{each \textit{value} in \Call{ORDER-DOMAIN-VALUES}{var, assignment, csp}} 
\If{value is consistent with assignment} 
\State add \{\textit{var} = value\} to assignment 
\State inferences $\leftarrow$ \Call{INERENCE}{csp, var, value} 
\If{inferences $\neq$ failure} 
\State add inferences to assignment 
\EndIf 
\EndIf 
\EndFor 
\State result $\leftarrow$ BACKTRACK(assignment, csp) 
\If{result $\neq$ failure} 
\State \textbf{return} result 
\EndIf 
\State remove \{\textit{var} = value\} and inferences from assignment 
\EndIf 
\State \textbf{return} failure 
\end{algorithmic}

\caption{A simple backtracking algorithm for constraint satisfaction problems. The algorithm is modeled on the recursive depth-first search of Chapter ??.
By varying the functions \textit{SELECT-UNASSIGNED-VARIABLE} and \textit{ORDER-DOMAIN-VALUES}, we can implement the general-purpose heuristics discussed in the text. The function \textit{INERENCE} can optionally be used to impose arc-, path-, or \textit{k}-consistency, as desired. If a value choice leads to failure (noticed either by \textit{INERENCE} or by \textit{BACKTRACK}), then value assignments (including those made by \textit{INERENCE}) are removed from the current assignment and a new value is tried.}

\caption{The \textit{MIN-CONFLICTS} algorithm for solving CSPs by local search. The initial state may be chosen randomly or by a greedy assignment process that chooses a minimal-conflict value for each variable in turn. The \textit{CONFLICTS} function counts the number of constraints violated by a particular value, given the rest of the current assignment.}

\end{document}
Example (2)
Example (3)
Example (4)
Improving Efficiency: CSP Heuristics & Pruning Techniques

- **Variable ordering**: Which one to assign first?
- **Value ordering**: Which value to try first?
- **Try to detect failures** early on
- **Try to exploit problem structure**

→ **Note**: all this is not problem-specific!
Variable Ordering:
Most constrained first

- Most constrained variable:
  - choose the variable with the fewest remaining legal values
  → reduces branching factor!
Variable Ordering: Most Constraining Variable First

- Break ties among variables with the same number of remaining legal values:
  - choose variable with the most constraints on remaining unassigned variables
  - reduces branching factor in the next steps
Value Ordering:
Least Constraining Value First

- Given a variable,
  - choose first a value that rules out the fewest values in the remaining unassigned variables

→ We want to find an assignment that satisfies the constraints (of course, this does not help if the given problem is unsatisfiable.)
Rule out Failures early on:
Forward Checking

- Whenever a value is assigned to a variable, values that are now illegal for other variables are removed.
- Implements what the ordering heuristics implicitly compute.
- $WA = \text{red}$, then $NT$ cannot become $\text{red}$.
- If all values are removed for one variable, we can stop!
Forward Checking (1)

- Keep track of remaining values
- Stop if all have been removed
Forward Checking (2)

- Keep track of remaining values
- Stop if all have been removed
Forward Checking (3)

- Keep track of remaining values
- Stop if all have been removed
Forward Checking (4)

- Keep track of remaining values
- Stop if all have been removed
Forward Checking: Sometimes it Misses Something

- Forward Checking propagates information from assigned to unassigned variables.
- However, there is no propagation between unassigned variables.
A directed arc $X \rightarrow Y$ is “consistent” iff
- for every value $x$ of $X$, there exists a value $y$ of $Y$, such that $(x, y)$ satisfies the constraint between $X$ and $Y$

Remove values from the domain of $X$ to enforce arc-consistency

Arc consistency detects failures earlier

Can be used as preprocessing technique or as a propagation step during backtracking
Arc Consistency Example
AC-3 Algorithm

**function** AC-3(*csp*) **returns** false if an inconsistency is found and true otherwise

**inputs:** *csp*, a binary CSP with components (*X*, *D*, *C*)

**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** REVISE(*csp*, *X_i*, *X_j*) **then**

**if** size of *D_i* = 0 **then** **return** false

**for each** *X_k* in *X_i*.NEIGHBORS - \{*X_j*\} **do**

add (*X_k*, *X_i*) to *queue*

**return** true

**function** REVISE(*csp*, *X_i*, *X_j*) **returns** true iff we revise the domain of *X_i*

revised ← false

**for each** *x* in *D_i* **do**

**if** no value *y* in *D_j* allows (*x*,*y*) to satisfy the constraint between *X_i* and *X_j* **then**

delete *x* from *D_i*

revised ← true

**return** revised

Figure 6.3
The arc-consistency algorithm AC-3. After applying AC-3, either every arc is arc-consistent, or some variable has an empty domain, indicating that the CSP cannot be solved. The name "AC-3" was used by the algorithm’s inventor (?) because it’s the third version developed in the paper.
Properties of AC-3

- AC-3 runs in $O(d^3n^2)$ time, with $n$ being the number of nodes and $d$ being the maximal number of elements in a domain.
- Of course, AC-3 does not detect all inconsistencies (which is an NP-hard problem).
CSP has two independent components
Identifiable as connected components of constraint graph
Can reduce the search space dramatically
If the CSP graph is a tree, then it can be solved in $O(nd^2)$ (general CSPs need in the worst case $O(d^n)$).

Idea: Pick root, order nodes, apply arc consistency from leaves to root, and assign values starting at root.
Pick any variable as root; choose an ordering such that each variable appears after its parent in the tree.

Apply arc-consistency to \((x_i, x_k)\) when \(x_i\) is the parent of \(x_k\) for all \(k = n\) down to 2 (any tree with \(n\) nodes has \(n - 1\) arcs and per arc \(d^2\) comparisons are needed, which results in a complexity of \(O(n \cdot d^2)\)).

Now we can start at \(x_1\) assigning values from the remaining domains without creating any conflict in one sweep through the tree!

This algorithm is linear in \(n\).
Problem Structure (3): Almost Tree-structured

Idea: Reduce the graph structure to a tree by fixing values in a reasonably chosen subset

Instantiate a variable and prune values in neighboring variables is called **Conditioning**
Algorithm **Cutset Conditioning:**

1. Choose a subset $S$ of the CSPs variables such that the constraint graph becomes a tree after removal of $S$. The set $S$ is called a **cycle cutset**.

2. For each possible assignment of variables in $S$ that satisfies all constraints on $S$
   - remove from the domains of the remaining variables any values that are inconsistent with the assignments for $S$, and
   - if the remaining CSP has a solution, return it together with the assignment for $S$

**Note:** Finding the smallest cycle cutset is NP hard, but several efficient approximation algorithms are known.
Another Method:
Tree Decomposition (1)

- Decompose the problem into a set of connected sub-problems, where two sub-problems are connected when they share a constraint
- Solve the sub-problems independently and then combine the solutions
Another Method:
Tree Decomposition (2)

- A tree decomposition must satisfy the following conditions:
  - Every variable of the original problem appears in at least one sub-problem
  - Every constraint appears in at least one sub-problem
  - If a variable appears in two sub-problems, it must appear in all sub-problems on the path between the two sub-problems
  - The connections form a tree
Another Method: Tree Decomposition (3)

- Consider sub-problems as new mega-variables, which have values defined by the solutions to the sub-problems.
- Use technique for tree-structured CSP to find an overall solution (constraint is to have identical values for the same variable).

\[
\begin{array}{l}
\{WA=red, NT=green, SA=blue\} \\
\{WA=red, NT=blue, SA=green\} \\
\{WA=blue, NT=green, SA=red\} \\
\vdots \\
\vdots \\
\{NT=blue, SA=green, Q=red\} \\
\{NT=green, SA=red, Q=blue\} \\
\{NT=green, SA=blue, Q=red\} \\
\vdots \\
\vdots
\end{array}
\]
The aim is to make the subproblems as small as possible. The tree width \( w \) of a tree decomposition is the size of largest sub-problem minus 1.

Tree width of a graph is minimal tree width over all possible tree decompositions.

If a graph has tree width \( w \) and we know a tree decomposition with that width, we can solve the problem in \( O(nd^{w+1}) \).

Unfortunately, finding a tree decomposition with minimal tree width is NP-hard. However, there are heuristic methods that work well in practice.
CSPs are a special kind of search problem:
- states are value assignments
- goal test is defined by constraints

Backtracking = DFS with one variable assigned per node. Other intelligent backtracking techniques possible

Variable/value ordering heuristics can help dramatically

Constraint propagation prunes the search space

Path-consistency is a constraint propagation technique for triples of variables

Tree structure of CSP graph simplifies problem significantly

Cutset conditioning and tree decomposition are two ways to transform part of the problem into a tree

CSPs can also be solved using local search