Artificial Intelligence Methods

Marc Erich Latoschik
AI & VR Lab
Artificial Intelligence Group
University of Bielefeld

Constraint Satisfaction Problems

Chapter 5
Section 1 – 3
Outline

• Constraint Satisfaction Problems (CSP)
• Backtracking search for CSPs
• Local search for CSPs

Constraint satisfaction problems (CSPs)

• Standard search problem:
  • state is a "black box" – any data structure that supports successor function, heuristic function, and goal test
• CSP:
  • state is defined by variables $X_i$ with values from domain $D_i$
  • goal test is a set of constraints specifying allowable combinations of values for subsets of variables

• Simple example of a formal representation language

• Allows useful general purpose algorithms with more power than standard search algorithms
Example: Map-Coloring

- **Variables** $WA, NT, Q, NSW, V, SA, T$
- **Domains** $D_i = \{\text{red, green, blue}\}$
- **Constraints**: adjacent regions must have different colors
  - e.g., $WA \neq NT$, or $(WA, NT)$ in $\{(\text{red, green}), (\text{red, blue}), (\text{green, red}), (\text{green, blue}), (\text{blue, red}), (\text{blue, green})\}$

**Solutions** are complete and consistent assignments, e.g., $WA = \text{red}$, $NT = \text{green}$, $Q = \text{red}$, $NSW = \text{green}$, $V = \text{red}$, $SA = \text{blue}$, $T = \text{green}$
Constraint graph

- **Binary CSP**: each constraint relates two variables
- **Constraint graph**: nodes are variables, arcs are constraints

Varieties of CSPs

- **Discrete variables**
  - finite domains:
    - $n$ variables, domain size $d \rightarrow O(d^n)$ complete assignments
    - e.g., Boolean CSPs, incl.~Boolean satisfiability (NP-complete)
  - infinite domains:
    - integers, strings, etc.
    - e.g., job scheduling, variables are start/end days for each job
    - need a constraint language, e.g., $\text{StartJob}_1 + 5 \leq \text{StartJob}_3$

- **Continuous variables**
  - e.g., start/end times for Hubble Space Telescope observations
  - linear constraints solvable in polynomial time by linear programming
Varieties of constraints

- **Unary** constraints involve a single variable,
  - e.g., $SA \neq \text{green}$

- **Binary** constraints involve pairs of variables,
  - e.g., $SA \neq WA$

- **Higher-order** constraints involve 3 or more variables,
  - e.g., cryptarithmetic column constraints

Example: Cryptarithmetic

Variables: $F, T, U, W$
Domains: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
Constraints:
- $\text{Alldiff}(F, T, U, W, R, O)$
- $O + O = R + 10 \cdot X_1$
- $X_1 + W + W = U + 10 \cdot X_2$
- $X_2 + T + T = O + 10 \cdot X_3$
- $X_3 = F, T \neq 0, F \neq 0$
Real-world CSPs

- Assignment problems
  - e.g., who teaches what class
- Timetabling problems
  - e.g., which class is offered when and where?
- Transportation scheduling
- Factory scheduling

- Notice that many real-world problems involve real-valued variables

Standard search formulation (incremental)

Let's start with the straightforward 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
  - fail if no legal assignments
- **Goal test**: the current assignment is complete

1. This is the same for all CSPs
2. Every solution appears at depth $n$ with $n$ variables
   - use depth-first search
3. Path is irrelevant, so can also use complete-state formulation
4. $b = (n - l)d$ at depth $l$, hence $n! \cdot d^n$ leaves
Backtracking search

- Variable assignments are **commutative**, i.e.,
  \[ WA = red \text{ then } NT = green \] same as \[ NT = green \text{ then } WA = red \]

- Only need to consider assignments to a single variable at each node
  \[ \rightarrow b = d \text{ and there are } d^n \text{ leaves} \]

- 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 a solution, or failure
    return Recursive-Backtracking(\{\}, csp)

function Recursive-Backtracking(assignment, csp) returns a solution, or failure
    if assignment is complete then return assignment
    var \leftarrow \text{SELECT-UNASSIGNED-VARIABLE}(\text{Variables}(csp), assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do:
        if value is consistent with assignment according to Constraints[csp] then
            add \{ var = value \} to assignment
            result \leftarrow \text{Recursive-Backtracking}(assignment, csp)
            if result \neq failure then return result
            remove \{ var = value \} from assignment
        return failure
```
Backtracking example
Backtracking example
Improving backtracking efficiency

- **General-purpose** methods can give huge gains in speed:
  - Which variable should be assigned next?
  - In what order should its values be tried?
  - Can we detect inevitable failure early?