Constraint satisfaction problems

Lirong Xia

Rensselaer Polytechnic Institute
Today’s schedule

.Constraint satisfaction problems

.Algorithms

• backtracking search
• filtering
  ▪ forward checking
  ▪ constraint propagation
Constraint Satisfaction Problems

- Standard search problems:
  - State is a “black box”: arbitrary data structure
  - Goal test: any function over states
  - Successor function can be anything

- Constraint satisfaction problems (CSPs):
  - A special subset of search problems
  - State is defined by variables $X_i$ with values from a domain $D$ (sometimes $D$ depends on $i$)
  - Goal test is a set of constraints specifying allowable combinations of values for subsets of variables

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

- **Variables:** WA, NT, Q, NSW, V, SA, T
- **Domains:** $D = \{\text{red, green, blue}\}$
- **Constraints:** adjacent regions must have different colors
  
  $WA \neq NT$

  $(WA, NT) \in \{(\text{red, green}), (\text{red, blue}), (\text{green, red}), \ldots\}$

- **Solutions are assignments satisfying all constraints,** e.g.:

  \[
  \begin{cases}
  WA = \text{red}, NT = \text{green}, Q = \text{red}, NSW = \text{green}, \\
  V = \text{red}, SA = \text{blue}, T = \text{green}
  \end{cases}
  \]
Example: N-Queens

- Formulation 1:
  - Variables: $X_{ij}$
  - Domains: $\{0,1\}$
  - Constraints
    \[
    \forall i, j, k \quad (X_{ij}, X_{ik}) \in \{(0, 0), (0, 1), (1, 0)\}
    \]
    \[
    \forall i, j, k \quad (X_{ij}, X_{kj}) \in \{(0, 0), (0, 1), (1, 0)\}
    \]
    \[
    \forall i, j, k \quad (X_{ij}, X_{i+k, j+k}) \in \{(0, 0), (0, 1), (1, 0)\}
    \]
    \[
    \forall i, j, k \quad (X_{ij}, X_{i+k, j-k}) \in \{(0, 0), (0, 1), (1, 0)\}
    \]
    \[
    \sum_{i,j} X_{ij} = N
    \]
Example: N-Queens

Formulation 2:

• Variables: $Q_k$
• Domains: $\{1,2,3,\ldots,N\}$
• Constraints

Implicit: $\forall i, j$ non-threatening $(Q_i, Q_j)$
-or-
Explicit: $(Q_1, Q_2) \in \{(1, 3), (1, 4), \ldots\}$
\ldots \ldots
**Constraint Graphs: Goal test**

- Binary CSP: each constraint relates (at most) two variables
- Binary constraint graph: nodes are variables, arcs show constraints
- General-purpose CSP algorithms use the graph structure to speed up search.
  - E.g., Tasmania is an independent subproblem!
Example: Cryptarithmetic

- **Variables (circles):**
  
  \[ F \ T \ U \ W \ R \ O \ X_1 \ X_2 \ X_3 \]

- **Domains:**
  
  \[ \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \]

- **Constraints (boxes):**

  \[
  \text{alldiff}(F, T, U, W, R, O) \\
  O + O = R + 10 \cdot X_1 \\
  \ldots
  \]

  \[
  \begin{array}{ccc}
  \text{T} & \text{W} & \text{O} \\
  + & \text{T} & \text{W} & \text{O} \\
  \hline
  \text{F} & \text{O} & \text{U} & \text{R}
  \end{array}
  \]
Example: Sudoku

- Variables:
  - Each (open) square

- Domains:
  - \( \{1, 2, \ldots, 9\} \)

- Constraints:
  - 9-way alldiff for each column
  - 9-way alldiff for each row
  - 9-way alldiff for each region
Varieties of CSPs

Discrete Variables:

- Finite domains
  - Size $d$ means $O(d^n)$ complete assignments
  - E.g., Boolean CSPs, including Boolean satisfiability (NP-complete)
- Infinite domains (integers, strings, etc.)
  - Linear constraints solvable

Continuous variables:

- Linear constraints solvable in polynomial time by LP methods
Varieties of Constraints

- **Variables of Constraints**
  - **Unary constraints** involve a single variable (equiv. to shrinking domains):
    \[ SA \neq \text{green} \]
  - **Binary constraints** involve pairs of variables:
    \[ SA \neq WA \]
  - **Higher-order constraints** involve 3 or more variables:
    e.g., cryptarithmetic column constraints

- **Preferences (soft constraints)**:
  - E.g., red is better than green
  - Often representable by a cost for each variable assignment
  - Gives constrained optimization problems
Standard Search Formulation

- Standard search formulation of CSPs (incremental)
- 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
  - Goal test: the current assignment is complete and satisfies all constraint
Search Methods

- What would BFS do?
- What would DFS do?
- Is BFS better or DFS better?
- Can we use A*?
Early detection of non-Goal
Backtracking Search

- Idea 1: only consider a single variable at each point
  - Variable assignments are commutative, so fix ordering
  - I.e., [WA = red then NT = green] same as [NT = green then WA = red]
  - Only need to consider assignments to a single variable at each step

- Idea 2: only allow legal assignments at each point
  - “Incremental goal test”

- DFS for CSPs with these two improvements is called backtracking search

- Backtracking search is the basic uninformed algorithm for CSPs

- Can solve n-queens for $n \approx 25$
Backtracking Example
Improving Backtracking

- Ordering:
  - Which variable should be assigned next?
  - In what order should its values be tried?
- Filtering: Can we detect inevitable failure early?
Minimum Remaining Values

- Minimum remaining values (MRV):
  - Choose the variable with the fewest legal values

- Why min rather than max?

- Also called “most constrained variable”

- “Fail-fast” ordering
Example

- For every variable, keep track of which values are still possible

only one possibility for last column; might as well fill in

now only one left for other two columns

done! (no real branching needed!)
Choosing a value

Given a choice of variable:

- Choose the least constraining value
- The one that rules out the fewest values in the remaining variables
- May take some computation to determine this!

Why least rather than most?
Filtering: Forward Checking

- Idea: keep track of remaining values for unassigned variables (using immediate constraints)
- Idea: terminate when any variable has no legal values
Filtering: Constraint Propagating

- Forward checking propagates information from assigned to unassigned variables, but doesn’t provide early detection for all failures:

  Why didn’t we detect this yet?

- NT and SA cannot both be blue!

  Why didn’t we detect this yet?
Consistency of An Arc

An arc $X \rightarrow Y$ is consistent iff for every $x$ in the tail there is some $y$ in the head which could be assigned without violating a constraint.

Forward checking = Enforcing consistency of each arc pointing to the new assignment.

Delete from tail!
Arc Consistency of a CSP

- A simple form of propagation makes sure all arcs are consistent:
  - If V loses a value, neighbors of V need to be rechecked!
  - Arc consistency detects failure earlier than forward checking
  - Can be run as a preprocessor or after each assignment
  - Might be time-consuming