Constraint Satisfaction Problems
Các bài toán thỏa mãn ràng buộc
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.
  - Aim is to find an assignment of $X_i$ from domain $D_i$ in such a way that none of the constraints are violated.

- 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 $D_i = \{\text{red}, \text{green}, \text{blue}\}$
- Constraints: adjacent regions must have different colors

- e.g., $WA \neq NT$, or $(WA,NT)$ in $\{(\text{red,green}), (\text{red,blue}), (\text{green,red}), (\text{green,blue}), (\text{blue,red}), (\text{blue,green})\}$
Example: Map-Coloring

- Solutions are complete and consistent assignments, e.g., WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green
Example: n-queens puzzle

- Assume one queen in each column.
- Variables $Q_1, ..Q_n$.
- Domains $D_i=\{1,..,n\}$
- Constraints
- $Q_i \neq Q_j$ (cannot be in the same row)
- $|Q_i - Q_j| \neq |i-j|$ (or same diagonal)
Example Sudoku
Real-world CSPs

• Assignment problems (e.g. who teaches what class)
• Timetabling problems (e.g. which class is offered when and where?)
• Hardware configuration
• Transport scheduling
• Factory scheduling
Constraint graph

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

- **Unary** constraints involve a single variable,
  - e.g., SA ≠ green
- **Binary** constraints involve pairs of variables,
  - e.g., SA ≠ WA
- **Higher-order** constraints involve 3 or more variables,
  - e.g., cryptarithmetic column constraints
- **Soft constraints** (preferences)
  - 11am lecture is better than 8am lecture
Example: Cryptarithmetic

- Variables: $F T U W R O X_1 X_2 X_3$
- Domains: $\{0,1,2,3,4,5,6,7,8,9\}$
- Constraints: $\text{Alldiff} (F,T,U,W,R,O)$
  - $O + O = R + 10 \cdot X_1$
  - $X_1 + W + W = U + 10 \cdot X_2$
  - $X_2 + T + T = O + 10 \cdot X_3$
  - $X_3 = F$, $T \neq 0$, $F \neq 0$
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 ($d$: number of variable values)
Backtracking search

• Variable assignments are *commutative*, i.e.,
  \[[ \text{WA} = \text{red} \text{ then } \text{NT} = \text{green} ] \text{ same as } [ \text{NT} = \text{green} \text{ then } \text{WA} = \text{red} ]\]
• Only need to consider assignments to a single variable at each node
  \[ b = d \text{ and there are } d^n \text{ leaves} \]
• 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 ← SELECT-UNASSIGNED-VARIABLE(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 ← 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

- **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?
Most constrained variable
Biến bị ràng buộc nhiều nhất

- Most constrained variable: choose the variable with the fewest legal values

- a.k.a. minimum remaining values (MRV) heuristic
Most constraining variable
Bien ràng bước nhiều nhất

• Tie-breaker among most constrained variables
• Most constraining variable (degree heuristic):
  – choose the variable with the most constraints on remaining variables
Least constraining value
Giá trị ràng buộc ít nhất

• 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
Forward checking
Kiểm tra trước

• **Idea:**
  – Keep track of remaining legal values for unassigned variables
  – Terminate search when any variable has no legal values
Forward checking

• Idea:
  – Keep track of remaining legal values for unassigned variables
  – Terminate search when any variable has no legal values
Forward checking

• Idea:
  – Keep track of remaining legal values for unassigned variables
  – Terminate search when any variable has no legal values
Forward checking

• **Idea:**
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values
Constraint propagation

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

- NT and SA cannot both be blue!
- **Constraint propagation** repeatedly enforces constraints locally
Arc consistency

- 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

- 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

• 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

Phạm Bảo Sơn
Arc consistency

- 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

**AC-3**

```plaintext
function AC-3(csp) returns the CSP, possibly with reduced domains
inputs: csp, a binary 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 RM-INCONSISTENT-VALUES(X_i, X_j) then
        for each X_k in Neighbors[X_i] do
            add (X_k, X_i) to queue

function RM-INCONSISTENT-VALUES(X_i, X_j) returns true iff remove a value
removed ← false
for each x in DOMAIN[X_i] do
    if no value y in DOMAIN[X_j] allows (x, y) to satisfy constraint(X_i, X_j)
        then delete x from DOMAIN[X_i]; removed ← true

return removed
```

- **Time complexity:** \(O(n^2d^3)\)
Special constraints

• Arc-consistency does miss some cases
• Example:
  – \{WA=red, NSW=red\}
  – AC-3: Domain for SA, NT, Q : \{green, blue\}
  – *Alldiff* constraint is violated as number of values is less than number of variables.
Local search for CSPs

- Local search or iterative improvement.
- Hill-climbing, simulated annealing typically work with "complete" states, i.e., all variables assigned.
- To apply to CSPs:
  - allow states with unsatisfied constraints
  - operators **reassign** variable values
- Variable selection: randomly select any conflicted variable
- Value selection by **min-conflicts (mẫu thuận ít nhất)** heuristic:
  - choose value that violates the fewest constraints
  - i.e., hill-climb with \( h(n) = \text{total number of violated constraints} \)
Example: 4-Queens

- **States:** 4 queens in 4 columns \((4^4 = 256\) states)  
- **Actions:** move queen in column  
- **Goal test:** no attacks  
- **Evaluation:** \(h(n) = \) number of attacks
Phase transition in CSP’s

- Given random initial state, can solve $n$-queens in almost constant time for arbitrary $n$ with high probability (e.g., $n = 10,000,000$)
- In general, randomly-generated CSP tend to be easy if there are very few or very many constraints. They become extra hard in a narrow range of the ratio:
Flat regions and local optima

- Sometimes, have to go sideways or even backwards in order to make progress towards the actual solution.
Simulated Annealing

• Stochastic hill climbing based on difference between evaluation of previous state ($h_0$) and new state ($h_1$).
• If $h_1 < h_0$, definitely make the change.
• Otherwise, make the change with probability:
  $$e^{-(h_1-h_0)/T},$$
  $T$ is a “temperature” parameter
• Reduces to ordinary hill climbing when $T=0$.
• Become totally random search as $T->\infty$
• We gradually decrease the value of $T$ during the search.
Summary

• CSPs are a special kind of problem:
  – states defined by values of a fixed set of variables
  – goal test defined by constraints on variable values
• Backtracking = depth-first search with one variable assigned per node
• Variable ordering and value selection heuristics help significantly
• Forward checking prevents assignments that guarantee later failure
• Constraint propagation (e.g., arc consistency) does additional work to constrain values and detect inconsistencies
• Iterative min-conflicts is usually effective in practice
• Simulated Annealing can help to escape from local optima.
References

• Artificial Intelligence, A modern approach. Chapter 5.