Artificial Intelligence Foundation – JC3001

Lecture 15: Constraint Satisfaction Problems III

Prof. Aladdin Ayesh (aladdin.ayesh@abdn.ac.uk)

Dr. Binod Bhattarai (binod.bhattarai@abdn.ac.uk)

Dr. Gideon Ogunniye, (g.ogunniye@abdn.ac.uk)

September 2025

Material adapted from:

Russell and Norvig (AIMA Book): Chapter 6

Course Progression

- Part 1: Introduction
  1. Introduction to AI ✓
  2. Agents ✓
- Part 2: Problem-solving
  1. Search 1: Uninformed Search ✓
  2. Search 2: Heuristic Search ✓
  3. Search 3: Local Search ✓
  4. Search 4: Adversarial Search ✓
- Part 3: Reasoning and Uncertainty
  1. **Reasoning 1: Constraint Satisfaction**
  2. Reasoning 2: Logic and Inference
  3. Probabilistic Reasoning 1: BNs
  4. Probabilistic Reasoning 2: HMMs
- Part 4: Planning
  1. Planning 1: Intro and Formalism
  2. Planning 2: Algos and Heuristics
  3. Planning 3: Hierarchical Planning
  4. Planning 4: Stochastic Planning
- Part 5: Learning
  1. Learning 1: Intro to ML
  2. Learning 2: Regression
  3. Learning 3: Neural Networks
  4. Learning 4: Reinforcement Learning
- Part 6: Conclusion
  1. Ethical Issues in AI
  2. Conclusions and Discussion

Objectives

- Defining Constraint Satisfaction Problems (CSP) ✓
- CSP examples ✓
- Backtracking search for CSPs ✓
- Local search for CSPs
- Problem structure and problem decomposition

Outline

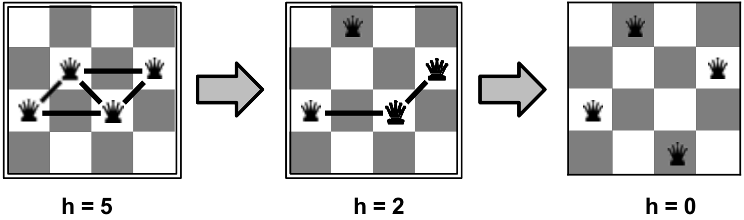
- Local Search for CSPs

- Local search algorithms can be very effective in solving many CSPs.
- Local search algorithms use a complete-state formulation where:
  - each state assigns a value to every variable, and
  - the search changes the value of one variable at a time.
- **Min-conflicts heuristic:** value that results in the minimum number of conflicts with other variables that brings us closer to a solution.
  - Usually has a series of plateaus
- **Plateau search:** allowing sideways moves to another state with the same score.
  - can help local search find its way off the plateau.
- **Constraint weighting** aims to concentrate the search on the important constraints
  - Each constraint is given a numeric weight, initially all 1.
  - weights adjusted by incrementing when it is violated by the current assignment

Min-Conflicts Search

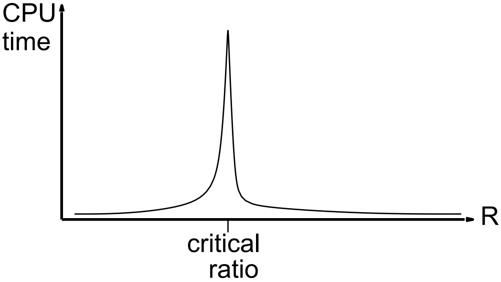
function Min-Conflicts(csp, max_steps) **returns** a solution or $failure$
    $current \leftarrow$ an initial complete assignment for csp
    for $i = 1$ to max_steps **do**
        if $current$ is a solution for csp **then return** $current$
        $var \leftarrow$ a randomly chosen conflicted variable from csp.Variables
        $value \leftarrow \operatorname*{arg\,min}_{v \in var.\text{Domain}}$ Conflicts$(csp, var, v, current)$
        $current.var \leftarrow value$
    return $failure$

- **States:** 4 queens in 4 columns ($4^4 = 256$ states)
- **Operators:** move queen in column
- **Goal test:** no attacks
- **Evaluation:** $h(n) =$ number of attacks

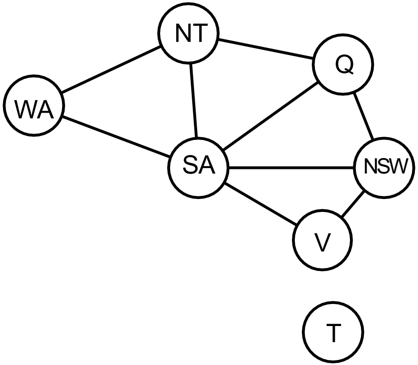


- Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., $n = 10,000,000$)
- The same appears to be true for any randomly-generated CSP **except** in a narrow range of the ratio:

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$



Problem structure



- Tasmania and mainland are **independent subproblems**
- Identifiable as **connected components** of constraint graph

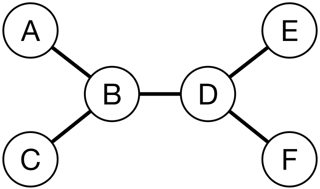
Problem structure (cont.)

- Suppose each subproblem has c variables out of n total
- Worst-case solution cost is $n/c \cdot d^c$, linear in n

E.g. $n = 80, d = 2, c = 20$

$2^{80} = 4$ billion years at 10 million nodes/sec

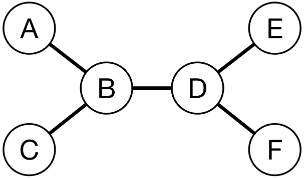
$4 \cdot 2^{20} = 0.4$ seconds at 10 million nodes/sec

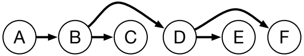


- **Theorem:** if the constraint graph has no loops, the CSP can be solved in $O(nd^2)$ time
- Compare to general CSPs, where worst-case time is $O(d^n)$
- This property also applies to logical and probabilistic reasoning:
  an important example of the relation between syntactic restrictions and the complexity of 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

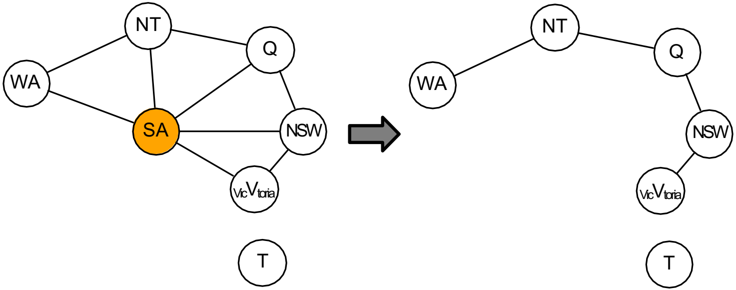




2. For j from n down to 2, apply RemoveInconsistent$(\text{Parent}(X_j), X_j)$
3. For j from 1 to n, assign X_j consistently with Parent(X_j)

Nearly tree-structured CSPs

Conditioning: instantiate a variable, prune its neighbous' 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

Iterative algorithms for CSPs

- 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** heuristic:
  - choose value that violates the fewest constraints
  - i.e., hillclimb with $h(n) =$ total number of violated constraints

- 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
- Local search using the min-conflicts heuristic has also been applied to constraint satisfaction problems with great success
- The CSP representation allows analysis of problem structure
- Tree-structured CSPs can be solved in linear time
- CSPs still **beat the most advanced ML methods** for most optimization tasks

Lecture Summary

- Modelling Constraint Satisfaction Problems
  - Solving CSP via Search
  - Heuristics
  - Solving CSP via Local Search

Any Questions.