CSE 473: Artificial Intelligence

Constraint Satisfaction Problems II
Factored (aka Structured) Search

What is Search For?

- **Planning**: sequences of actions
  - The *path to the goal* is the important thing
  - Paths have various costs, depths
  - Assume little about problem structure

- **Identification**: assignments to variables
  - The *goal itself* is important, *not the path*
  - All paths at the same depth (for some formulations)
Constraint Satisfaction Problems

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

- Making use of CSP formulation allows for optimized algorithms
  - Typical example of trading generality for utility (in this case, speed)

---

CSP Example: Map Coloring

- **Variables:** WA, NT, Q, NSW, V, SA, T
- **Domains:** $D = \{\text{red, green, blue}\}$
- **Constraints:** adjacent regions must have different colors
  - Implicit: WA $\neq$ NT
  - Explicit: $(\text{WA, NT}) \in \{(\text{red, green}), (\text{red, blue}), \ldots\}$
- **Solutions are assignments satisfying all constraints, e.g.:**
  - $\{\text{WA=red, NT=green, Q=red, NSW=green, V=red, SA=blue, T=green}\}$
Systematic Search to Solve CSP

- **States** – partial assignments to variables
- **Operators** – assign another variable
- **Initial State** – no variables assigned
- **Goal State** – all vars assigned & constraints satisfied

We’ll improve this basic method to exploit structure

Backtracking Search

- Backtracking search is the basic uninformed algorithm for solving CSPs
- **Start with Depth First Search**
  - “backtracking search” IS a Kind of depth first search with these 2 details:
- **Idea 1: One variable at a time**
  - 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: Check constraints as you go**
  - I.e. consider only values which do not conflict previous assignments
  - Might have to do some computation to check the constraints
  - “Incremental goal test”
- Can solve n-queens for \( n \approx 25 \)
Backtracking Example

Backtracking Search

```plaintext
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({}, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/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 given 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
```

- What are the choice points?

[Demo: coloring -- backtracking]
Improving Backtracking

- General-purpose ideas give huge gains in speed
- Ordering:
  - Which variable should be assigned next?
  - In what order should its values be tried?
- Filtering: Can we detect inevitable failure early?
- Structure: Can we exploit the problem structure?

Filtering
Filtering: Forward Checking

- Filtering: Keep track of domains for unassigned variables and cross off bad options
- Forward checking: Cross off values that violate a constraint when added to the existing assignment

[Demo: coloring -- forward checking]

Video of Demo Coloring – Backtracking with Forward Checking
Filtering: Constraint Propagation

- Forward checking only propagates information from assigned to unassigned
- It doesn't catch when **two unassigned variables** have no consistent assignment:
  - NT and SA cannot both be blue!
  - Why didn't we detect this yet?
  - Constraint propagation: reason from constraint to constraint

Consistency of a Single 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 arcs pointing to each new assignment
Arc Consistency of an **Entire CSP**

- A simple form of propagation makes sure all arcs are consistent:

  ![Arc Consistency Diagram](image)

  - Important: 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
  - What’s the **downside** of enforcing arc consistency?

---

**AC-3 algorithm for Arc Consistency**

```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 Remove-Inconsistent-Values(X_i, X_j) then
    for each X_k in Neighbors[X_i] do
      add (X_k, X_i) to queue

function Remove-Inconsistent-Values(X_i, X_j) returns true iff succeeds
removed ← false
for each x in Domain[X_i] do
  if no value y in Domain[X_j] allows (x, y) to satisfy the constraint X_i \rightarrow X_j
  then delete x from Domain[X_i]; removed ← true
return removed
```

- Runtime: \(O(n^2d^3)\), can be reduced to \(O(n^2d^2)\)
- ... but detecting **all** possible future problems is NP-hard – why?

[Demo: CSP applet (made available by aispace.org) -- n-queens]
Limitations of Arc Consistency

- After enforcing arc consistency:
  - Can have one solution left
  - Can have multiple solutions left
  - Can have no solutions left (and not know it)

- Arc consistency still runs inside a backtracking search!

[Demo: coloring -- forward checking]
[Demo: coloring -- arc consistency]