L02: Constraint Satisfaction

AIMA4e: Chapter 6.1–5

What you should know after this lecture

- CSP solution strategies:
  - Backtracking
  - Forward checking
  - Learning within a problem (if we have time)
  - Local search
- Constraint graphs

Constraint-satisfaction problem: formal definition

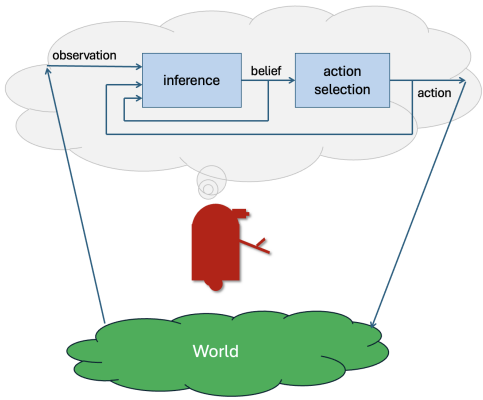
- $\mathcal{X}$ is a set of variables $\{X_1, \ldots, X_n\}$
- $\mathcal{D}$ is a set of domains $\{D_1, \ldots, D_n\}$, where $D_i = \{x_1, \ldots, x_k\}$ is the set of possible values of X_i
- $\mathcal{C}$ is a set of constraints:
  - *scope* : a tuple of variables
  - *relation* : a relation specifying tuples of values that this tuple of variables can legally take on

Define:

- *assignment* : mapping from variables to values
- *partial assignment*: only provides values for some variables
- *consistent assignment* : partial assignment that doesn't violate any constraints
- *solution* : complete assignment that doesn't violate any constraints

CSPs: factored belief representation

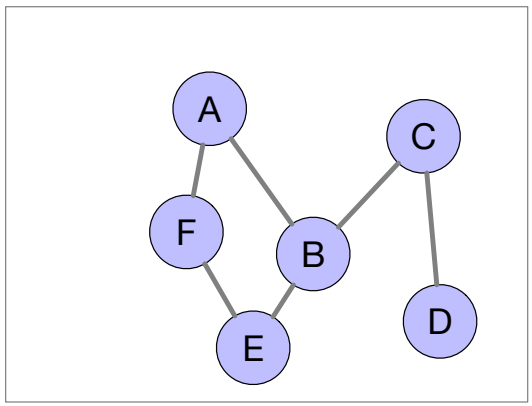
- Set-based belief : $B \subset \mathcal{S}$
- $B = \{(x^1, \ldots, x^n) \text{ for } x^i \in D^i \mid \text{consistent}((x^1, \ldots, x^n), \mathcal{C})\}$
- Each time we add a constraint (get new information), we reduce the set of possible world states.
- We might be interested in the possible set of values of X_i, after all the constraints have been taken into account.

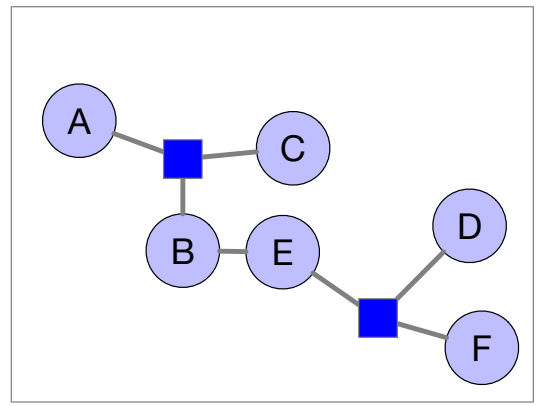


Constraints

- **Unary** constraint only involves a single variable: use to reduce the domain of that variable
- **Binary** constraint involves two variables (domains can be any size). We will focus on binary constraints. Discussion in book and HW problem on reducing higher-order constraints to binary, and other ways of handling them.

Constraint (hyper)graph: useful to visualize constraint structure





If it does not contain any loops, then there's a cool, efficient message passing algorithm.

Factory problem: smaller

Objects:

- Machines: sander, painter, dryer
- Parts: A, B
- Times: $1, \ldots, 4$

Constraints:

- Each part must be sanded before painted before dried.
- The sander and painter can each operate on at most one part at a time.
- The sander can't operate at the same time the dryer is operating.

One formulation: Variables: pm: when is part p in machine m? Domain of variables are times.

Hey robot, where are my keys?

- Rooms: bedroom, bathroom, office, kitchen
- Objects: keys, laptop, wallet, purse

Constraints:

- My keys are in the same place as my wallet.
- My wallet is not in my purse.
- My purse is next to my laptop.
- My laptop is in my office.
- I would never put my wallet in the bathroom.

Where should the robot look?

Backtracking (= depth-first search)

a is an assignment, initially $\{\}$

```
BACKTRACK(a)
    if COMPLETE(a): return a
    X = UNASSIGNED-VAR(a)
    for x ∈ DOMAIN-VALUES(X):
        if CONSISTENT(a, {X = x}):
            EXTEND(a, {X = x})
            r = BACKTRACK(a)
            if r ≠ 'failed': return r // Save r and continue to generate
            REMOVE(a, {X = x})
    return 'failed'
```

Is this better than the stupidest possible algorithm?

Variable and value ordering

Dynamically, during search:

- **variables**: UNASSIGNED-VAR chooses the variable with the fewest values in its domain
- **values**: DOMAIN-VALUES orders values earlier that rule out the fewest choices for variables it's connected to in the constraint graph

These will be especially useful in combination with some inference methods.

Basic arc consistency: inference method

REVISE(X_i, X_j, C_{ij})
Given:

- Two variables: X_i and X_j with domains: D_i and D_j
- Constraint C_{ij}

$D_i := \{x_i \in D_i \mid \exists x_j \in D_j . (x_i, x_j) \in C_{ij}\}$

Removes values from domain of X_i that are inconsistent with values of domain of X_j.
Easy to extend to take $\mathcal{C}$ as argument and pick out relevant constraints.

Usually, for efficiency, we implement with side effects. Be careful to undo!

BT with forward checking

```
FC(X, a):
    for X_i ∈ (UNASSIGNED-VAR(a) ∩ NEIGHBORS(X))
        REVISE(X_i, X)
        if D_i = { }: return 'failed'

BACKTRACK-FC(a)
    if COMPLETE(a): return a
    X = UNASSIGNED-VAR(a)
    for x ∈ DOMAIN-VALUES(X):
        if CONSISTENT(a, {X = x}):
            EXTEND(a, {X = x})
            r = FC(X, a)
            if r ≠ 'failed':
                r = BACKTRACK(a)
                if r ≠ 'failed': return r
            REMOVE(a, {X = x}); UNDO-FC(X, a)
    return 'failed'
```

Arc consistency

Can work harder to be sure that all arcs are consistent.

- See AC-3 alg in book.
- Roughly, keep doing REVISE until no domains change further.
- Completely solves some problems.
- BT-AC3 often more expensive than BT-FC.
- Can extend the idea to making k-tuples (for $k > 2$) of variables consistent.

Backjumping

Sometimes we have made a poor initial choice, but end up with endlessly considering assignments to irrelevant variables.

- Whenever a dead-end occurs at variable X, backtrack to the "most recent" variable that is connected to X in the constraint graph.
- Can be very helpful!

Requires careful bookkeeping to be sure all the right assignments and inferences are undone.

AIMA4e asserts that any assignment that is pruned by backjumping will also be pruned by forward-checking. Prove it to yourself!

Learning while searching

Idea: find assignments that are no good: not simply inconsistent themselves, but such that there is no possible way to assign the rest of the variables.

- conflict set for a variable X: Set of variables X' and values x' such that there is no assignment to X consistent with $X' = x'$. It's minimal if no subset of it is a conflict set.
- Once you discover a conflict set, don't ever try it again!
- Add a constraint that forbids this assignment and keep going. (But note that it's non-binary).

Identifying and recording only conflict sets which are known to be minimal constitutes deep learning. – Dechter, AIJ, 1990

Local search: a very different strategy!

- Start with a complete assignment, with constraint viols
- Until you reach a satisfying assignment: pick a variable and assign a new value.

Guidance helps! **Min-conflicts heuristic**:

- Randomly choose a variable that is in conflict (violating some constraint)
- Assign it the value that will minimize the total number of constraints violated.

Simulated annealing:

- Propose a move (variable and value) at random.
- If it reduces the number of conflicts, accept it.
- If it does not, accept anyway, with probability $e^{-\Delta/T}$ where Δ is number of conflicts added and T is a temperature parameter that is decreased over time.

Min conflict not guaranteed to find solution; simulated annealing is (eventually)

Message passing

When your constraint (hyper)graph is a tree (has no loops) then there's a super-cool algorithm!

- Pick any node to be root
- Construct a topological sort: every node is in the list after its parent.
- Starting at the *end* of the list, do, for each X REVISE(parent(X), X)
- Each X is left with a domain such that any value in the remaining domain is consistent with the whole subtree beneath it.
- After this O(n) processing, select any value at root, and work forward selecting any consistent value. No backtracking needed.

But! What if you don't have a tree?

Three strategies:

- Run the variable elimination algorithm, which runs in exponential time in the "tree width" of the constraint graph
- Make a tree by combining some variables into super-variables with the product of their domains and do message-passing.
- Find a cutset: a set of variables, such that if they were removed, the remaining (hyper)graph would be a tree.
  - Do backtracking on values of the variables in the cutset
  - Given an assignment to those variables, do message-passing to try to find assignment to the rest.

We will see these algorithms again in probabilistic inference!

Next time

- Probabilistic graphical models (factor graphs, in particular) are a generalization of CSPs!