L03: Constraint Satisfaction

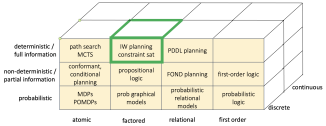
AIMA4e: Chapter 6

Pseudocode in these notes is informal. See text for more details.

What you should know after this lecture

- Definition of a constraint-satisfaction problem
- Solution strategies:
  - Backtracking
  - Forward checking and fancier consistency
  - Learning within a problem
  - Local search
- How to formulate a CSP

Factored states and information



Factored, discrete states
Factored "observations" as constraints

Inference about state based on certain observations

- State space is factored into a set of state variables
- Observations are constraints on (pieces of information about) the values of those state variables
- Our objective is to figure out one or more possible states that are consistent with the observations.

Inference doesn't increase our information about the underlying state—just processes it into a more useful form

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

CSP Objectives

Three possible objectives:

1. Solution: find a satisfying assignment or prove one does not exist.
2. All solutions: find all satisfying assignments.
3. Inference: conclude what values a variable must have.

Mostly we focus on solution but worth keeping the others in mind.

CSP is NP-Complete: time exponential in domain size in worst case

Constraint-satisfaction problems: context

Why study CSP?

- A CSP formulation exposes structure in the problem that enables efficient inference
- There are many professional, efficient CSP solvers
- There are lots of important problems that can be formulated as CSP
- So, if you can formulate as a CSP, it can often be solved efficiently

Factory problem

Objects:

- Machines: sander, painter, dryer
- Parts: $p_1, \ldots, p_3$
- Times: $1, \ldots, 5$

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.

How can we formulate this as a CSP?

Variations on the theme

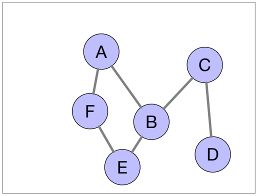
Theme: fixed number of variables with finite discrete domains
Variations:

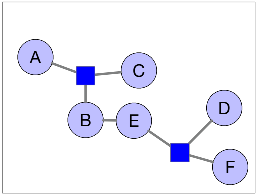
- Infinite discrete domains
- Continuous domains
  - With linear constraints == <u>linear programming</u>
    Polynomial time in number of variables!
  - Some other classes have good specialized solutions (e.g., quadratic programs)
- Getting <u>all</u> solutions
- Assigning costs and finding least cost solution == <u>constrained optimization</u>

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.

Stupidest possible algorithm

```
STUPID
    for each possible assignment $A$:
        if $A$ does not violate any $C \in \mathcal{C}$:
            return $A$
    return 'failed'
```

- How many A are there?
- Can we do better in the worst case?
- Can we do better in many cases?

Backtracking (= depth-first search)

a is an assignment, initially { }

```
BACKTRACK($a$)
    if COMPLETE($a$): return $a$
    $X$ = UNASSIGNED-VAR($a$)
    for $x \in$ DOMAIN-VALUES($X$):
        if CONSISTENT($a, \{X = x\}$):
            EXTEND($a, \{X = x\}$)
            $r$ = BACKTRACK($a$)
            if $r \neq$ 'failed': return $r$
            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.

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

Forward checking

```
FC($X, a$):
    for $X_i \in$ UNASSIGNED-VAR($a$) $\cap$ NEIGHBORS($X$)
        REVISE($X_i, X$)
        if $D_i = \{\ \}$: return 'failed'

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

Arc consistency

Can work harder to be sure that <u>all</u> 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!