Chapter 5

Constraint satisfaction problems (CSPs)

A standard search problem is a "black box"—any old data structure that supports goal test, heuristic, successor functions. CSPs differ because:

- CSPs have a formal representation language
- CSPs are defined by variables, values, domains, and constraints. E.g.,

```
domains: D_i = \{red, green, blue\}
constraints:
- WA \neq NT
- WA \neq N.S.W
- WA \neq Q
- WA \neq V
- WA \neq T
- N.T \neq N.S.W
- N.T \neq Q
- N.T \neq V
- N.T \neq T
- N.S.W \neq Q
- N.S.W \neq V
- N.S.W \neq T
- Q \neq V
- Q \neq T
- V \neq T
```

The representation language of CSPs allows for a more powerful and expressive way of describing problems. CSPs can represent problems that would be difficult or impossible to express in a standard search problem.

Example: Map-coloring

A map-coloring problem can be represented as a CSP with variables for each region and constraints specifying that adjacent regions must have different colors.

Solution: Assigning the colors red, green, and blue to the regions in a way that satisfies all constraints.

Example: Map-coloring contd.

```
W.A = red, N.T = green, Q = red, N.S.W = green, V = red, S.A = blue, T = green
```

Constraint graph

In a binary CSP, each constraint relates at most two variables. The constraint graph visualizes the relationships between variables and constraints.

Example: Map-coloring

The constraint graph for the map-coloring problem shows the relationships between variables and constraints in a visual way.

Outline

- CSP examples
- Backtracking search for CSPs
- Problem structure and problem decomposition
- Local search for CSPs
- CSP algorithms use the graph structure
Varieties of CSPs

Discrete variables
- finite domains; size $d \Rightarrow O(d^n)$ complete assignments
  - e.g., Boolean CSPs, incl. Boolean satisfiability (NP-complete)
- infinite domains (integers, strings, etc.)
  - e.g., job scheduling, variables are start/end days for each job
  - need a constraint language, e.g.,
    \[
    \text{StartJob}_1 + 5 \leq \text{StartJob}_3
    \]
  - linear constraints solvable, nonlinear undecidable

Continuous variables
- e.g., start/end times for Hubble Telescope observations
- linear constraints solvable in poly time by LP methods

Chapter 5

Varieties of constraints

Unary constraints involve a single variable, e.g.,
\[ \text{SA} \not= \text{green} \]

Binary constraints involve pairs of variables, e.g.,
\[ \text{SA} \not= \text{WA} \]

Higher-order constraints involve 3 or more variables

Preferences (soft constraints), e.g.,
\[ \text{red} \text{is better than green} \]

often representable by a cost for each variable assignment
\[ \Rightarrow \text{constrained optimization problems} \]

Chapter 5

Example: Cryptarithmetic

Variables:
\[ F, O, T, U, W, R, O, X \]

Domains:
\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}

Constraints
\[ \text{alldiff}(F, T, U, W, R, O) \]
\[ O + O = R + 10 \cdot X \]
\[ \text{where} W \text{ is the carry, } W, O \in \{0, 1\} \]

Chapter 5

Real-world CSPs

Assignment problems
- e.g., who teaches what class

Timetabling problems
- e.g., which class is offered when and where?

Hardware configuration

Transportation scheduling

Factory scheduling

Floorplanning

Notice that many real-world problems involve real-valued variables

Chapter 5

Standard search formulation (incremental)

Let's start with the straightforward, dumb approach, then fix it if

\begin{align*}
\text{Initial state} & : \text{the empty assignment, } \{\} \\
\text{Successor function} & : \text{assign a value to an unassigned variable} \\
\text{Goal test} & : \text{the current assignment is complete} \\
\text{Failed if no legal assignments (not fixable)} \\
\end{align*}

\[ \Rightarrow \text{fail if no legal assignments (not fixable)} \]

\[ \Rightarrow \text{fix if possible} \]

\[ \Rightarrow \text{constrained optimization problems} \]

- often represented by a cost for each variable assignment

- hard to solve

- higher-order constraints involve 3 or more variables

- many constraints involve more than 1 variable

- unary constraints involve a single variable

Chapter 5

Example: Cryptarithmetic

Variables:
\[ F, O, T, U, W, R, O, X \]

Domains:
\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}

Constraints
\[ \text{alldiff}(F, T, U, W, R, O) \]
\[ O + O = R + 10 \cdot X \]

Chapter 5

Varieties of CSPs

Linear constraints solvable by LP methods

integer constraints solved by poly time LP methods

integer, linear, submodular, fractional

satisfiable by linear programming

While dominoes are easy, k-Color is NP-complete

integer constraints solved by poly time LP methods

integer, linear, submodular, fractional

satisfiable by linear programming

While dominoes are easy, k-Color is NP-complete
Let's start with the straightforward, dumb approach, then fix it.

**States** are defined by the values assigned so far:

- **Initial state**: the empty assignment, \( \emptyset \).
- **Successor function**: assign a value to an unassigned variable that does not conflict with the current assignment.

⇒ fail if no legal assignments (not fixable!).

- **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) \( b = \binom{n}{d} \), at depth \( d \), \( n \) leaves!!!

4) Path is irrelevant, so can also use complete-state formulation.

**Backtracking search**:

Variable assignments are **commutative**, i.e.,

\[
\text{\[ W A = \text{red} \Rightarrow N T = \text{green} \] same as \[ N T = \text{green} \Rightarrow W A = \text{red} \]}
\]

Only need to consider assignments to a single variable at each node.

⇒ \( b = d \) and there are \( d \) leaves at depth \( d \)."
Backtracking search

Backtracking example

Improving backtracking efficiency

Improving backtracking efficiency

Backtracking example

Backtracking example

Improving backtracking efficiency

Backtracking example

Backtracking example

Backtracking example

Backtracking example

General-purpose methods can give huge gains in speed:

1. Which variable should be assigned next?
2. In what order should its values be tried?
3. Can we detect inevitable failure early?
4. Can we use the advantages of problem structure?
Minimum remaining values (MRV):
choose the variable with the fewest legal values

Tie-breaker among MRV variables
Degree heuristic:
choose the variable with the most constraints on remaining variables
Seems simple (and is) but is still best method for k-colouring.

Least constraining value
choose a variable that has 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
Idea: keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values

Degree heuristic:
Idea: keep track of remaining legal values for unassigned variables
Tie-breaker among MRV variables
Choose the variable with the most constraints on remaining variables
Seems simple (and is) but is still best method for k-colouring.
Forward checking

Idea: Keep track of remaining values for each variable that need to be assigned.

For every variable, if there is no legal value, then forward checking is violated.

Simplest form of propagation makes each arc consistent.

Arc consistency

Constraint propagation

Simplest form of propagation makes each arc consistent. For every variable, if there is some value not in the domain, then forward checking is violated.

Forward checking propagates information from assigned variables to unassigned variables. Constraint propagation terminates search when any value has no legal values where it can be assigned.

Arc consistency

If variable X loses a value, neighbors of X need to be rechecked.

Termination search when any variable has no legal values.

Forward checking propagates information from assigned variables to unassigned variables.

Arc consistency

If variable X loses a value, neighbors of X need to be rechecked.

Termination search when any variable has no legal values.

Forward checking propagates information from assigned variables to unassigned variables.

Arc consistency

If variable X loses a value, neighbors of X need to be rechecked.

Termination search when any variable has no legal values.

Forward checking propagates information from assigned variables to unassigned variables.

Arc consistency

If variable X loses a value, neighbors of X need to be rechecked.

Termination search when any variable has no legal values.
Arc consistency

The simplest form of propagation makes each arc consistent.

\[ X \rightarrow Y \text{ is consistent iff for every value of } X \text{ there is some allowed } y. \]

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

```plaintext
Chapter 5.37

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
    \( (X_i, X_j) \leftarrow \text{Remove-First}(queue) \)
    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 \( \leftarrow \) false
for each \( x \) in Domain[\( X_i \)] do
    if no value \( y \) in Domain[\( X_j \)] allows \( (x, y) \) to satisfy the constraint \( X_i \leftrightarrow X_j \) then
        delete \( x \) from Domain[\( X_i \)]; removed \( \leftarrow true \)
return removed
```

Complexity:

\[ O(n^2d^3), \text{ can be reduced to } O(n^2d^2) (\text{but detecting all is NP-hard}) \]

Problem structure contd.

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., \( n = 80 \), \( d = 2 \), \( c = 20 \) gives \( 4 \text{ billion years at } 10 \text{ million nodes/sec} \).

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

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.

The CSP representation allows analysis of problem structure.

Tree-structured CSPs can be solved in linear time.

Iterative min-conflicts is usually effective in practice.