Outline of Today’s Class

CSP Examples

Backtracking Search for CSPs

Problem Structure and Problem Decomposition

Local Search for CSPs

Take-home Problem

CSP Summary
Constraint Satisfaction Problems (CSPs)

Standard search problem:
**state** is a “black box”—any old data structure that supports goal test, eval, successor

CSP:
**state** is defined by variables $X_i$ with values from domain $D_i$
**goal test** is a set of constraints specifying allowable combinations of values for subsets of variables

Simple example of a **formal representation language**

Allows useful **general-purpose** algorithms with more power than standard search algorithms
Example: Map-Coloring

Variables $WA, NT, Q, NSW, V, SA, T$

Domains $D_i = \{\text{red}, \text{green}, \text{blue}\}$

Constraints: adjacent regions must have different colors

- $WA \neq NT$ (if the language allows this), or
- $(WA, NT) \in \{(\text{red}, \text{green}), (\text{red}, \text{blue}), (\text{green}, \text{red}), (\text{green}, \text{blue}), \ldots\}$
Solutions are assignments satisfying all constraints, e.g.,
\[\{WA = \text{red}, NT = \text{green}, Q = \text{red}, NSW = \text{green}, V = \text{red}, SA = \text{blue}, T = \text{green}\}\]
Constraint Graph

Binary CSP: each constraint relates at most two variables

Constraint graph: nodes are variables, arcs show constraints

General-purpose CSP algorithms use the graph structure to speed up search. E.g., Tasmania is an independent subproblem!
Varieties of CSPs

Discrete variables

- finite domains; size $d \implies O(d^n)$ 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 polynomial time by linear programming (LP)
Varieties of Constraints

Unary constraints involve a single variable
e.g., $SA \neq green$

Binary constraints involve pairs of variables
e.g., $SA \neq WA$

Higher-order constraints involve 3 or more variables
e.g., cryptarithmetic column constraints

Strong vs. soft constraints

Preferences (soft constraints)
e.g., red is better than green
often representable by a cost for each variable assignment
→ constrained optimization problems
Example: Cryptarithmetic

Variables: $F \ T \ U \ W \ R \ O \ X_1 \ X_2 \ X_3$

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

Constraints

- $\text{alldiff}(F, T, U, W, R, O)$
- $O + O = R + 10 \cdot X_1$, etc.
Real-world CSPs

Assignment problems
e.g., who teaches what class

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

Hardware configuration

Spreadsheets

Transportation scheduling

Factory scheduling

Floorplanning

Real-world problems almost always involve real-valued variables
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 current assignment. $
\Rightarrow$ 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
   $\Rightarrow$ use depth-first search

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

4) $b = (n - \ell)d$ at depth $\ell$, hence $n!d^n$ leaves!!!! 😞
Variable assignments are **commutative**, i.e.,

\[
\begin{align*}
&W A = \text{red} \text{ then } N T = \text{green} \quad \text{same as} \quad [N T = \text{green} \text{ then } W A = \text{red}]
\end{align*}
\]

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

\[b = d \quad \text{and there are } d^n \text{ leaves}\]

Depth-first search for CSPs with single-variable assignments is called **backtracking** search

Backtracking search is the basic uninformed algorithm for CSPs

Can solve \(n\)-queens for \(n \approx 25\)
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
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 take advantage 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
Least Constraining Value

Given a variable, choose 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

**Idea**: Terminate search when any variable has no legal values
**Forward Checking**

**Idea:** Keep track of remaining legal values for unassigned variables

**Idea:** Terminate search when any variable has no legal values
**Idea:** Keep track of remaining legal values for unassigned variables

**Idea:** Terminate search when any variable has no legal values
**Idea:** Keep track of remaining legal values for unassigned variables

**Idea:** Terminate search when any variable has no legal values
Constraint Propagation

Forward checking propagates information from assigned to unassigned variables, but doesn’t provide early detection for all failures:

\[ NT \quad and \quad SA \quad cannot \quad both \quad be \quad blue! \]

Constraint propagation repeatedly enforces constraints locally
Arc Consistency

Simplest form of propagation makes each arc consistent

\( X \rightarrow Y \) is consistent iff

for every value \( x \) of \( X \) there is some allowed \( y \)
Arc Consistency

Simplest form of propagation makes each arc consistent

\( X \rightarrow Y \) is consistent iff

for every value \( x \) of \( X \) there is some allowed \( y \)
Arc Consistency

Simplest form of propagation makes each arc consistent

\( X \rightarrow Y \) is consistent iff

for every value \( x \) of \( X \) there is some allowed \( y \)

If \( X \) loses a value, neighbors of \( X \) need to be rechecked
Arc Consistency

Simplest form of propagation makes each arc consistent
X \rightarrow Y \text{ is consistent iff}

for every value x of X 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
function AC-3(\( \text{csp} \)) returns the CSP, possibly with reduced domains

**inputs:** \( \text{csp} \), a binary CSP with variables \( \{X_1, X_2, \ldots, X_n\} \)

**local variables:** \( \text{queue} \), a queue of arcs, initially all the arcs in \( \text{csp} \)

**while** \( \text{queue} \) is not empty **do**

\((X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\text{queue})\)

**if** \( \text{REMOVE-INCONSISTENT-VALUES}(X_i, X_j) \) **then**

**for each** \( X_k \) **in** \( \text{NEIGHBORS}[X_i] \) **do**

add \((X_k, X_i)\) to \( \text{queue} \)

**function** \( \text{REMOVE-INCONSISTENT-VALUES}(X_i, X_j) \) **returns** true iff succeeds

\( \text{removed} \leftarrow \text{false} \)

**for each** \( x \) **in** \( \text{DOMAIN}[X_i] \) **do**

**if** no value \( y \) **in** \( \text{DOMAIN}[X_j] \) allows \((x,y)\) to satisfy the constraint \( X_i \leftrightarrow X_j \) **then** delete \( x \) **from** \( \text{DOMAIN}[X_i] \)

\( \text{removed} \leftarrow \text{true} \)

**return** \( \text{removed} \)
Given: $c$ constraints, $\leq d$ values in the domain of each variable $X_i$

How many $(X_k, X_i)$ arcs will be added to the queue when pruning domain of some $X_i$?
Given: $c$ constraints, $\leq d$ values in the domain of each variable $X_i$

How many $(X_k, X_i)$ arcs will be added to the queue when pruning domain of some $X_i$? At most $\text{deg}(X_i)$
Time Complexity Arc Consistency Algorithm

Given: $c$ constraints, $\leq d$ values in the domain of each variable $X_i$

How many $(X_k, X_i)$ arcs will be added to the queue when pruning domain of some $X_i$?
   at most $\text{deg}(X_i)$

How many is this over all variables?

How long does it take to check consistency of an arc?
   $O(d^2)$

So, putting it all together:
   $T(\text{AC-3}) \in O(cd^3)$
Given: $c$ constraints, $\leq d$ values in the domain of each variable $X_i$

How many $(X_k, X_i)$ arcs will be added to the queue when pruning domain of some $X_i$? at most $\text{deg}(X_i)$

How many is this over all variables?
sum over all degrees is $O(E)$ of constraint graph
Given: $c$ constraints, $\leq d$ values in the domain of each variable $X_i$

How many $(X_k, X_i)$ arcs will be added to the queue when pruning domain of some $X_i$? at most $\text{deg}(X_i)$

How many is this over all variables?

- sum over all degrees is $O(E)$ of constraint graph
- which is $O(c)$
Time Complexity Arc Consistency Algorithm

Given: $c$ constraints, $\leq d$ values in the domain of each variable $X_i$

How many $(X_k, X_i)$ arcs will be added to the queue when pruning domain of some $X_i$?

at most $\text{deg}(X_i)$

How many is this over all variables?

sum over all degrees is $O(E)$ of constraint graph

which is $O(c)$

How often will the domain of each variable be pruned?

How long does it take to check consistency of an arc?

$O(d^2)$

So, putting it all together:

$T(AC^{-3}) \in O(cd^3)$
Given: \( c \) constraints, \( \leq d \) values in the domain of each variable \( X_i \)

How many \((X_k, X_i)\) arcs will be added to the queue when pruning domain of some \( X_i \)?
   at most \( \text{deg}(X_i) \)

How many is this over all variables?
   sum over all degrees is \( O(E) \) of constraint graph
   which is \( O(c) \)

How often will the domain of each variable be pruned?
   cannot be more than the actual size of the domain
Given: $c$ constraints, $\leq d$ values in the domain of each variable $X_i$

How many $(X_k, X_i)$ arcs will be added to the queue when pruning domain of some $X_i$?

at most $\text{deg}(X_i)$

How many is this over all variables?

sum over all degrees is $O(E)$ of constraint graph
which is $O(c)$

How often will the domain of each variable be pruned?

cannot be more than the actual size of the domain

...so... $O(d)$ times

In total, how many arcs $(X_k, X_i)$ will be added to the queue over all variables?

$O(cd)$

How long does it take to check consistency of an arc?

$O(d^2)$

So, putting it all together:

$T(\text{AC}) \in O(cd^3)$
Given: \( c \) constraints, \( \leq d \) values in the domain of each variable \( X_i \)

How many \((X_k, X_i)\) arcs will be added to the queue when pruning domain of some \( X_i \)?

at most \( \deg(X_i) \)

How many is this over all variables?

sum over all degrees is \( O(E) \) of constraint graph

which is \( O(c) \)

How often will the domain of each variable be pruned?

cannot be more than the actual size of the domain

...so... \( O(d) \) times

In total, how many arcs \((X_k, X_i)\) will be added to the queue over all variables?
Given: $c$ constraints, $\leq d$ values in the domain of each variable $X_i$

How many $(X_k, X_i)$ arcs will be added to the queue when pruning domain of some $X_i$?
  at most $\text{deg}(X_i)$

How many is this over all variables?
  sum over all degrees is $O(E)$ of constraint graph
  which is $O(c)$

How often will the domain of each variable be pruned?
  cannot be more than the actual size of the domain
  ...so... $O(d)$ times

In total, how many arcs $(X_k, X_i)$ will be added to the queue over all variables?
  $O(cd)$
Time Complexity Arc Consistency Algorithm

Given: $c$ constraints, $\leq d$ values in the domain of each variable $X_i$

How many $(X_k, X_i)$ arcs will be added to the queue when pruning domain of some $X_i$?
   at most $\text{deg}(X_i)$

How many is this over all variables?
   sum over all degrees is $O(E)$ of constraint graph
   which is $O(c)$

How often will the domain of each variable be pruned?
   cannot be more than the actual size of the domain
   ...so... $O(d)$ times

In total, how many arcs $(X_k, X_i)$ will be added to the queue over all variables?
   $O(cd)$

How long does it take to check consistency of an arc?

Amarda Shehu (580)
Given: $c$ constraints, $\leq d$ values in the domain of each variable $X_i$

How many $(X_k, X_i)$ arcs will be added to the queue when pruning domain of some $X_i$? at most $\text{deg}(X_i)$

How many is this over all variables?
- sum over all degrees is $O(E)$ of constraint graph
- which is $O(c)$

How often will the domain of each variable be pruned?
- cannot be more than the actual size of the domain
  ...so... $O(d)$ times

In total, how many arcs $(X_k, X_i)$ will be added to the queue over all variables? $O(cd)$

How long does it take to check consistency of an arc? $O(d^2)$
Given: \( c \) constraints, \( \leq d \) values in the domain of each variable \( X_i \)

How many \((X_k, X_i)\) arces will be added to the queue when pruning domain of some \( X_i \)?
   at most \( \text{deg}(X_i) \)

How many is this over all variables?
   sum over all degrees is \( O(E) \) of constraint graph
   which is \( O(c) \)

How often will the domain of each variable be pruned?
   cannot be more than the actual size of the domain
   ...so... \( O(d) \) times

In total, how many arces \((X_k, X_i)\) will be added to the queue over all variables?
   \( O(cd) \)

How long does it take to check consistency of an arc?
   \( O(d^2) \)

So, putting it all together: \( T(\text{AC} - 3) \in O(cd^3) \)
Time Complexity Arc Consistency Algorithm

Given: $c$ constraints, $\leq d$ values in the domain of each variable $X_i$

How many $(X_k, X_i)$ arces will be added to the queue when pruning domain of some $X_i$? at most $\text{deg}(X_i)$

How many is this over all variables? sum over all degrees is $O(E)$ of constraint graph which is $O(c)$

How often will the domain of each variable be pruned? cannot be more than the actual size of the domain ...so... $O(d)$ times

In total, how many arcs $(X_k, X_i)$ will be added to the queue over all variables? $O(cd)$

How long does it take to check consistency of an arc? $O(d^2)$

So, putting it all together: $T(\text{AC} - 3) \in O(cd^3)$
Tasmania and mainland are independent subproblems

Identifiable as connected components of constraint graph
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 $O(n d^2)$ 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 REMOVE_INCONSISTENT($\text{Parent}(X_j), X_j$)

3. For $j$ from 1 to $n$, assign $X_j$ consistently with $\text{Parent}(X_j)$
Nearly Tree-structured CSPs

**Conditioning:** instantiate a variable, prune its neighbors’ domains

![Diagram showing the process of conditioning in a tree-structured CSP](image)

**Cutset conditioning:** instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree

Cutset size $c \implies$ 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., hill-climber with $h(n) =$ total number of violated constraints

**Take-home**: Propose a simple EA for 4-queens CSP
Example: 4-Queens as CSP

States:
Example: 4-Queens as CSP

**States:** 4 queens in 4 columns ($4^4 = 256$ states)
**Example: 4-Queens as CSP**

**States:** 4 queens in 4 columns \((4^4 = 256 \text{ states})\)

**Operators:**
Example: 4-Queens as CSP

**States:** 4 queens in 4 columns \(4^4 = 256\) states

**Operators:** move queen in column
States: 4 queens in 4 columns \((4^4 = 256\) states)

Operators: move queen in column

Goal test:
**Example: 4-Queens as CSP**

**States:** 4 queens in 4 columns \((4^4 = 256\) states)  

**Operators:** move queen in column  

**Goal test:** no attacks
Example: 4-Queens as CSP

**States:** 4 queens in 4 columns \(4^4 = 256\) states

**Operators:** move queen in column

**Goal test:** no attacks

**Evaluation:**
Example: 4-Queens as CSP

**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
Example: 4-Queens as CSP

**States:** 4 queens in 4 columns \(4^4 = 256 \text{ states}\)

**Operators:** move queen in column

**Goal test:** no attacks

**Evaluation:** \(h(n) = \text{number of attacks}\)

\[
\begin{align*}
\text{h = 5} & \quad \rightarrow \quad \text{h = 2} & \quad \rightarrow \quad \text{h = 0}
\end{align*}
\]
Performance of Min-conflicts

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 $R = \frac{\text{number of constraints}}{\text{number of variables}}$
Work through the 4-queens as CSP in greater detail

Assume one queen in each column. Which row does each one go in?
Work through the 4-queens as CSP in greater detail

Assume one queen in each column. Which row does each one go in?

**Variables** $Q_1, Q_2, Q_3, Q_4$
4-Queens as a CSP

Work through the 4-queens as CSP in greater detail

Assume one queen in each column. Which row does each one go in?

**Variables** \( Q_1, Q_2, Q_3, Q_4 \)

**Domains** \( D_i = \{1, 2, 3, 4\} \)
Work through the 4-queens as CSP in greater detail

Assume one queen in each column. Which row does each one go in?

**Variables** $Q_1, Q_2, Q_3, Q_4$

**Domains** $D_i = \{1, 2, 3, 4\}$

**Constraints**

- $Q_i \neq Q_j$ (cannot be in same row)
- $|Q_i - Q_j| \neq |i - j|$ (or same diagonal)
Work through the 4-queens as CSP in greater detail

Assume one queen in each column. Which row does each one go in?

**Variables** $Q_1, Q_2, Q_3, Q_4$

**Domains** $D_i = \{1, 2, 3, 4\}$

**Constraints**

- $Q_i \neq Q_j$ (cannot be in same row)
- $|Q_i - Q_j| \neq |i - j|$ (or same diagonal)

Translate each constraint into set of allowable values for its variables
Work through the 4-queens as CSP in greater detail

Assume one queen in each column. Which row does each one go in?

Variables $Q_1$, $Q_2$, $Q_3$, $Q_4$

Domains $D_i = \{1, 2, 3, 4\}$

Constraints

- $Q_i \neq Q_j$ (cannot be in same row)
- $|Q_i - Q_j| \neq |i - j|$ (or same diagonal)

Translate each constraint into set of allowable values for its variables

E.g., values for $(Q_1, Q_2)$ are $(1, 3)$ $(1, 4)$ $(2, 4)$ $(3, 1)$ $(4, 1)$ $(4, 2)$
CSPs are a special kind of search problems:
- states defined by values of a fixed set of variables
- goal test defined by \textit{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