Artificial Intelligence
Chapter 6: Constraint Satisfaction Problems
Digression: The Four Color Theorem

• One of the most famous results in the history of mathematics.

• It was first conjectured in 1852, but only finally proven in 1976. Notably it was the first math proof to rely crucially on computers (for a large set of configuration/case checks) – and for this reason was considered controversial.
Digression: The Four Color Theorem

• Why was the proof so difficult?

Because the best-known technique relied on (originally) 1936 unavoidable configurations -- like these.
Digression: The Four Color Theorem

• While we can’t prove the FCT in lecture (!), we can prove its baby brother, the 6 color theorem.
• Let’s do this…
• (1) Prove Euler’s Polyhedron Formula.
Digression: The Four Color Theorem

- (1) Prove Euler’s Polyhedron Formula. (we use induction on E(G)).
Digression: The Four Color Theorem

• (1) Prove Euler’s Polyhedron Formula.
• (2) Convince yourself that embedding on the sphere is equivalent to planarity.
Digression: The Four Color Theorem

• (1) Prove Euler’s Polyhedron Formula.
• (2) Convince yourself that embedding on the sphere is equivalent to planarity.

This is called a “stereographic projection”
Digression: The Four Color Theorem

• (1) Prove Euler’s Polyhedron Formula.
• (2) Convince yourself that embedding on the sphere is equivalent to planarity.
• (3) Prove that if $G$ is planar (with $n \geq 3$), then $|E(G)| \leq 3n-6$. 
Digression: The Four Color Theorem

- (1) Prove Euler’s Polyhedron Formula.
- (2) Convince yourself that embedding on the sphere is equivalent to planarity.
- (3) Prove that if $G$ is planar (with $n \geq 3$), then $|E(G)| \leq 3n-6$.
- (4) Claim: Every planar graph contains a vertex of degree 5 or less.
Digression: The Four Color Theorem

• (1) Prove Euler’s Polyhedron Formula.
• (2) Convince yourself that embedding on the sphere is equivalent to planarity.
• (3) Prove that if $G$ is planar (with $n \geq 3$), then $|E(G)| \leq 3n-6$.
• (4) Claim: Every planar graph contains a vertex of degree 5 or less. (Hint: use the fact that the sum of the degrees of vertices in a graph equals twice the number of edges).
Digression: The Four Color Theorem

- (1) Prove Euler’s Polyhedron Formula.
- (2) Convince yourself that embedding on the sphere is equivalent to planarity.
- (3) Prove that if $G$ is planar (with $n \geq 3$), then $|E(G)| \leq 3n - 6$.
- (4) Claim: Every planar graph contains a vertex of degree 5 or less.
- Now put it all together and we have the 6 Color Theorem!
Digression: The Four Color Theorem

- So how about a Five Color Theorem?
- Actually, it’s not *that* bad. We use the Six Color Theorem plus an additional clever argument utilizing a structure called a “Kempe Chain” (1890); the result follows by contradiction.
Digression: The Four Color Theorem

• Unfortunately, no one to date has found a simple way to reduce Kempe Chains and similar structures to efficiently solve 4CT (aside from exhaustive case-checking).

• Nevertheless, the 4CT and its proof serve evidence that humankind and machines can work together productively and harmoniously!
Digression: The Four Color Theorem

• Unfortunately, no one to date has found a simple way to reduce Kempe Chains and similar structures to efficiently solve 4CT (aside from exhaustive case-checking.

• Nevertheless, the 4CT and its proof serve evidence that humankind and machines can work together productively and harmoniously!

*This message sponsored by generous donations from your friends at:
Constraint satisfaction problems (CSPs)

• Standard search problem: state is a "black box“ – any data structure that supports successor function and goal test

• 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

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

- **Variables** $WA, NT, Q, NSW, V, SA, T$
- **Domains** $D_i = \{\text{red, green, blue}\}$
- **Constraints**: adjacent regions must have different colors
- e.g., $WA \neq NT$, or $(WA, NT)$ in $\{(\text{red, green}), (\text{red, blue}), (\text{green, red}), (\text{green, blue}), (\text{blue, red}), (\text{blue, green})\}$
Example: Map-Coloring

- Solutions are complete and consistent assignments
- e.g., WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green
Constraint graph

- **Binary CSP:** each constraint relates two variables
- **Constraint graph:** nodes are variables, arcs are constraints
Varieties of CSPs

• Discrete variables
  – finite domains:
    • $n$ variables, domain 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., $\text{StartJob}_1 + 5 \leq \text{StartJob}_3$

• Continuous variables
  – e.g., start/end times for Hubble Space Telescope observations
  – linear constraints solvable in polynomial time by LP
Varieties of constraints

- **Unary** constraints involve a single variable,
  - e.g., $SA \neq \text{green}$

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

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

• Try this one:
Cryptarithmetic Problem

Initial State

\[
\begin{array}{c}
\text{SEND} \\
+ \text{MORE} \\
\hline
\text{MONEY}
\end{array}
\]

- M = 1, S = 8 or 9
- O = 0 or 1 → O = 0
- N = E or E + 1 → N = E + 1
- C2 = 1, N + R > 8, E <> 9

- E = 2

- N = 3, R = 8 or 9
  - 2 + D = Y or 2 + D = 10 + Y
  - C1 = 0
  - C1 = 1

- 2 + D = Y
  - N + R = 10 + E
  - R = 9, S = 8

- D = 8
  - Y = 0
    - Conflict
  - D = 9
    - Y = 1
      - Conflict

- Solution:

\[
\begin{array}{c}
\text{SEND} \\
+ \text{MORE} \\
\hline
\text{MONEY}
\end{array}
\]

\[
\begin{array}{c}
9567 \\
+ 1085 \\
\hline
10652
\end{array}
\]

- A useful heuristic can help to select the best guess to try first.
- If there is a letter that participate in many constraints, then it is a good idea to prefer it to a letter that participates in a few.
Backtracking search

- Backtracking search is used for a depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign.

- It repeatedly chooses an unassigned variable, and then tries all values in the domain of that variable in turn, trying to find a solution.

- If an inconsistency is detected, then BACKTRACK returns failure, causing the previous call to try another value.
Backtracking Search

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

• Variable assignments are commutative, i.e.,
  \[ WA = \text{red then NT = green} \] same as \[ NT = \text{green then WA = red} \].

• \( \Rightarrow \) Only need to consider assignments to a single variable at each node, so the algorithm keeps only a single representation of a state and alters that representation rather than creating new ones.

• Can solve \( n \)-queens for \( n \approx 25 \).
Backtracking example
Backtracking example
Backtracking example
Backtracking example
Improving backtracking efficiency

• (3) General-purpose methods can give huge gains in speed:
  – Which variable should be assigned next? (e.g. MRV)
  – In what order should its values be tried? (e.g. inference/forward checking)
  – Can we detect inevitable failure early? (e.g. constraint learning)
Constraint Propagation

• In regular state-space search, an algorithm can do only one thing: *search*.

• In CSP there is a choice: (1) an algorithm can *search* (i.e. choose a new variable assignment from several possibilities); (2) perform a specific type of *inference* called *constraint propagation*. 

Constraint Propagation

• Constraint propagation uses the constraints to reduce the number of legal values for a variable, which in turn can reduce the legal values for another variable, and so on.

• Constraint propagation may be interleaved with search, or it can be done as a preprocessing step.

• The key idea is: local consistency.
Constraint Propagation

• The key idea is: **local consistency** (e.g. node consistency, arc consistency, etc.).

• If we treat each variable as a **node** in the graph, and each binary constraint as an **arc**, then the process of enforcing local consistency in each part of the graph causes inconsistent values to be eliminated throughout the graph.
Most constrained variable

- Most constrained variable:
  choose the variable with the fewest legal values

- a.k.a. **minimum remaining values (MRV)** heuristic
Most constraining variable

• A good idea is to use it as a tie-breaker among most constrained variables
• Most constraining variable:
  – choose the variable with the most constraints on remaining variables
Least constraining value

• Given a variable to assign, choose the least constraining value:
  – the one that rules out the fewest values in the remaining variables

• Combining these heuristics (MRV + LCV) makes 1000 queens feasible!
Forward checking

- MRV and LCV and related approaches can infer reduction in the domain variables \textit{before} we begin the search.

- But \textit{inference can be even more powerful in the course of a search}: every time we make a choice of a value for a variable, we have a brand-new opportunity to infer new domain reductions on neighboring values.
Forward checking

• Forward checking is one of the simplest forms of inference.

• Whenever a variable $X$ is assigned, the forward-checking process establishes **arc consistency** for it: for each unassigned variable $Y$ that is connected to $X$ by a constraint, delete from $Y$’s domain any value that is inconsistent with the value chosen for $X$. 
Forward checking

• **Idea:**
  – Keep track of remaining legal values for unassigned variables.
  – Terminate search when any variable has no legal values.
Forward checking

• Idea:
  – Keep track of remaining legal values for unassigned variables.
  – Terminate search when any variable has no legal values.
Forward checking

- Idea:
  - Keep track of remaining legal values for unassigned variables.
  - Terminate search when any variable has no legal values.
Forward checking

• **Idea:**
  – Keep track of remaining legal values for unassigned variables.
  – 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 and SA cannot both be blue!

- **Constraint propagation** algorithms repeatedly enforce constraints locally…
Node Consistency

• A single variable (in a CSP network) is **node-consistent** if all the values in the variable’s domain satisfy the variable’s unary constraints.

• E.g. Suppose South Australians dislike green; the variable SA starts with domain: \{red, green, blue\}, and we make it node-consistent by eliminating green, leaving SA with the reduced domain: \{red, blue\}.

• We say that a network is node-consistent if every variable in the network is node-consistent.
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 \textit{consistent}.

• $X \rightarrow Y$ is consistent iff
  \begin{align*}
  \text{for every value } x \text{ of } X \text{ there is some allowed } y
  \end{align*}

• 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.
AC-3 Algorithm

• The most popular algorithm for arc consistency is called AC-3.

• The AC-3 algorithm maintains a queue of arcs to consider.

• Initially, the queue contains all the arcs in the CSP. AC-3 the pops off an arbitrary arc \((X_i, X_j)\) from the queue and makes \(X_i\) arc-consistent with respect to \(X_j\).
AC-3 Algorithm

• If this leaves $D_i$ unchanged, the algorithm moves on to the next arc.
• If this revises $D_i$, then we add to the queue all arcs $(X_k, X_i)$, where $X_k$ is a neighbor of $X_i$.
• We need to do this because the change in $D_i$ might enable further reductions in the domains of $D_k$.
• Continue this process…
• We end up with a CSP equivalent to the original CSP – but the arc-consistent CSP will in most cases be much faster to search.
Arc consistency algorithm AC-3

```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
    \((X_i, X_j)\leftarrow\text{REMOVE-FIRST}(queue)\)
    if RM-INCONSISTENT-VALUES\((X_i, X_j)\) then
        for each \(X_k\) in NEIGHBORS[X_i] do
            add \((X_k, X_i)\) to queue

function RM-INCONSISTENT-VALUES\((X_i, X_j)\) returns true iff remove a value
removed\leftarrow false
for each \(x\) in DOMAIN[X_i] do
    if no value \(y\) in DOMAIN[X_j] allows \((x,y)\) to satisfy constraint\((X_i, X_j)\)
    then delete \(x\) from DOMAIN[X_i]; removed\leftarrow true
return removed
```

- Time complexity: \(O(\#\text{constraints} \cdot |\text{domain}|^3)\)

  Checking consistency of an arc is \(O(|\text{domain}|^2)\)
Path Consistency

• A two-variable set \( \{X_i, X_j\} \) is **path-consistent** with respect to a third variable \( X_m \), if, for every assignment \( \{X_i=a, X_j=b\} \) consistent with the constraints on \( \{X_i, X_j\} \), there is an assignment to \( X_m \) that satisfies the constraints on \( \{X_i, X_m\} \) and \( \{X_m, X_j\} \).

• This is called path-consistency, because one can think of it as looking at a path from \( X_i \) to \( X_j \) with \( X_m \) in the middle.
**k-consistency**

- A CSP is *k-consistent* if, for any set of k-1 variables, and for any consistent assignment to those variables, a consistent value can always be assigned to any kth variable.
- 1-consistency is node consistency.
- 2-consistency is arc consistency.
- For binary constraint networks, 3-consistency is the same as *path consistency*.
- Getting k-consistency requires time and space exponential in k.
- *Strong k-consistency* means k’-consistency for all k’ from 1 to k
  - Once strong k-consistency for k=#variables has been obtained, solution can be constructed trivially.
- Tradeoff between propagation and branching.
- Practitioners usually use 2-consistency and less commonly 3-consistency.
Other techniques for CSPs

• Global constraints
  – E.g., Alldiff
  – E.g., Atmost(10,P1,P2,P3), i.e., sum of the 3 vars ≤ 10
  – Special propagation algorithms
    • Bounds propagation
      – E.g., number of people on two flight D1 = [0, 165] and D2 = [0, 385]
      – Constraint that the total number of people has to be at least 420
      – Propagating bounds constraints yields D1 = [35, 165] and D2 = [255, 385]
    • ...

• Symmetry breaking (e.g. reduce search by imposing arbitrary ordering constraint).
Structured CSPs
Tree-structured CSPs

**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 $\text{REMOVEINCONSISTENT}(\text{Parent}(X_j), X_j)$

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

**Cutset conditioning:** instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree. (Finding the minimum cutset is NP-complete.)

Cutset size \(c\) \(\Rightarrow\) runtime \(O(d^c \cdot (n - c)d^2)\), very fast for small \(c\)
Tree decomposition

- Every variable in original problem must appear in at least one subproblem
- If two variables are connected in the original problem, they must appear together (along with the constraint) in at least one subproblem
- If a variable occurs in two subproblems in the tree, it must appear in every subproblem on the path that connects the two

- Algorithm: solve for all solutions of each subproblem. Then, use the tree-structured algorithm, treating the subproblem solutions as variables for those subproblems.
- O(nd^{w+1}) where w is the treewidth (= one less than size of largest subproblem)
- Finding a tree decomposition of smallest treewidth is NP-complete, but good heuristic methods exists