CS 188: Artificial Intelligence

Constraint Satisfaction Problems I

Instructor: Marco Alvarez

University of Rhode Island

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]
Constraint Satisfaction Problems
Constraint Satisfaction Problems

$N$ variables
Constraint Satisfaction Problems

$N$ variables

$x_1$
Constraint Satisfaction Problems

$N$ variables
Constraint Satisfaction Problems

$N$ variables

domain $D$
Constraint Satisfaction Problems

$N$ variables

domain $D$

constraints
Constraint Satisfaction Problems

\( N \) variables

domain \( D \)

constraints

states
Constraint Satisfaction Problems

N variables

domain D

constraints

states

goal test
Constraint Satisfaction Problems

N variables

domain D

constraints

states  goal test  successor function
Constraint Satisfaction Problems

N variables
domain D
constraints

states
partial assignment

goal test

successor function
Constraint Satisfaction Problems

N variables
domain D
constraints

states
partial assignment

goal test
complete; satisfies constraints

successor function
Constraint Satisfaction Problems

- $N$ variables
- domain $D$
- constraints

**States**
- partial assignment

**Goal Test**
- complete; satisfies constraints

**Successor Function**
- assign an unassigned variable
What is Search For?

- Assumptions about the world: a single agent, deterministic actions, fully observed state, discrete state space
What is Search For?

- Assumptions about the world: a single agent, deterministic actions, fully observed state, discrete state space

- Planning: sequences of actions
  - The path to the goal is the important thing
  - Paths have various costs, depths
  - Heuristics give problem-specific guidance
What is Search For?

- Assumptions about the world: a single agent, deterministic actions, fully observed state, discrete state space

- Planning: sequences of actions
  - The path to the goal is the important thing
  - Paths have various costs, depths
  - Heuristics give problem-specific guidance

- Identification: assignments to variables
  - The goal itself is important, not the path
  - All paths at the same depth (for some formulations)
  - CSPs are specialized for identification problems
Constraint Satisfaction Problems
Constraint Satisfaction Problems

- Standard search problems:
  - State is a “black box”: arbitrary data structure
  - Goal test can be any function over states
  - Successor function can also be anything
Constraint Satisfaction Problems

- Standard search problems:
  - State is a “black box”: arbitrary data structure
  - Goal test can be any function over states
  - Successor function can also be anything
Constraint Satisfaction Problems

- **Standard search problems:**
  - State is a “black box”: arbitrary data structure
  - Goal test can be any function over states
  - Successor function can also be anything

- **Constraint satisfaction problems (CSPs):**
  - A special subset of search problems
  - State is defined by variables $X_i$ with values from a domain $D$ (sometimes $D$ depends on $i$)
  - Goal test is a set of constraints specifying allowable combinations of values for subsets of variables
Constraint Satisfaction Problems

- **Standard search problems:**
  - State is a “black box”: arbitrary data structure
  - Goal test can be any function over states
  - Successor function can also be anything

- **Constraint satisfaction problems (CSPs):**
  - A special subset of search problems
  - State is defined by variables $X_i$ with values from a domain $D$ (sometimes $D$ depends on $i$)
  - Goal test is a set of constraints specifying allowable combinations of values for subsets of variables
Constraint Satisfaction Problems

- **Standard search problems:**
  - State is a “black box”: arbitrary data structure
  - Goal test can be any function over states
  - Successor function can also be anything

- **Constraint satisfaction problems (CSPs):**
  - A special subset of search problems
  - State is defined by variables $X_i$ with values from a domain $D$ (sometimes $D$ depends on $i$)
  - 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
CSP Examples

Western Australia
Northern Territory
South Australia
Queensland
New South Wales
Victoria
Tasmania
Example: Map Coloring
Example: Map Coloring

- Variables:
Example: Map Coloring

- Variables: WA, NT, Q, NSW, V, SA, T
Example: Map Coloring

- **Variables:** WA, NT, Q, NSW, V, SA, T
- **Domains:**

![Map Coloring Image]
Example: Map Coloring

- **Variables:** WA, NT, Q, NSW, V, SA, T
- **Domains:** \( D = \{\text{red, green, blue}\} \)
Example: Map Coloring

- Variables: WA, NT, Q, NSW, V, SA, T
- Domains: \( D = \{\text{red, green, blue}\} \)
- Constraints: adjacent regions must have different colors
Example: Map Coloring

- **Variables:** WA, NT, Q, NSW, V, SA, T
- **Domains:** $D = \{\text{red, green, blue}\}$
- **Constraints:** adjacent regions must have different colors
  
  Implicit: $WA \neq NT$
Example: Map Coloring

- **Variables:** WA, NT, Q, NSW, V, SA, T

- **Domains:** $D = \{\text{red, green, blue}\}$

- **Constraints:** adjacent regions must have different colors
  
  Implicit: $WA \neq NT$

  Explicit: $(WA, NT) \in \{(\text{red, green}), (\text{red, blue}), \ldots\}$
Example: Map Coloring

- **Variables:** WA, NT, Q, NSW, V, SA, T
- **Domains:** \( D = \{\text{red, green, blue}\} \)
- **Constraints:** adjacent regions must have different colors
  - Implicit: WA \(\neq\) NT
  - Explicit: \((WA, NT) \in \{(\text{red, green}), (\text{red, blue}), \ldots\}\)
- **Solutions are assignments satisfying all constraints, e.g.:**
Example: Map Coloring

- **Variables**: WA, NT, Q, NSW, V, SA, T
- **Domains**: \( D = \{\text{red, green, blue}\} \)
- **Constraints**: adjacent regions must have different colors
  - Implicit: WA \( \neq \) NT
  - Explicit: \((WA, NT) \in \{(\text{red, green}), (\text{red, blue}), \ldots\}\)
- **Solutions** are assignments satisfying all constraints, e.g.:
  \[
  \{WA=\text{red, NT=green, Q=red, NSW=green, V=red, SA=blue, T=green}\}
  \]
Example: N-Queens
Example: N-Queens
Example: N-Queens

- **Formulation 1:**
  - Variables: $X_{ij}$
  - Domains: $\{0, 1\}$
  - Constraints
Example: N-Queens

**Formulation 1:**
- **Variables:** $X_{ij}$
- **Domains:** $\{0, 1\}$
- **Constraints**

\[
\forall i, j, k \ (X_{ij}, X_{ik}) \in \{(0, 0), (0, 1), (1, 0)\}
\]
Example: N-Queens

- **Formulation 1:**
  - Variables: $X_{ij}$
  - Domains: $\{0, 1\}$
  - Constraints

\[
\forall i, j, k \ (X_{ij}, X_{ik}) \in \{(0, 0), (0, 1), (1, 0)\} \\
\forall i, j, k \ (X_{ij}, X_{kj}) \in \{(0, 0), (0, 1), (1, 0)\} \\
\forall i, j, k \ (X_{ij}, X_{i+k,j+k}) \in \{(0, 0), (0, 1), (1, 0)\} \\
\forall i, j, k \ (X_{ij}, X_{i+k,j-k}) \in \{(0, 0), (0, 1), (1, 0)\}
\]
Example: N-Queens

- **Formulation 1:**
  - Variables: $X_{ij}$
  - Domains: $\{0, 1\}$
  - Constraints

\[
\forall i, j, k \ (X_{ij}, X_{ik}) \in \{(0, 0), (0, 1), (1, 0)\} \\
\forall i, j, k \ (X_{ij}, X_{kJ}) \in \{(0, 0), (0, 1), (1, 0)\} \\
\forall i, j, k \ (X_{ij}, X_{i+k,j+k}) \in \{(0, 0), (0, 1), (1, 0)\} \\
\forall i, j, k \ (X_{ij}, X_{i+k,j-k}) \in \{(0, 0), (0, 1), (1, 0)\} \\
\sum_{i,j} X_{ij} = N
\]
Example: N-Queens

- **Formulation 2:**
  - **Variables:** $Q_k$
  - **Domains:** $\{1, 2, 3, \ldots N\}$
  - **Constraints:**
Example: N-Queens

- **Formulation 2:**
  - Variables: $Q_k$
  - Domains: $\{1, 2, 3, \ldots N\}$
  - Constraints:

    Implicit: $\forall i, j$ non-threatening($Q_i, Q_j$)
Example: N-Queens

- **Formulation 2:**
  - **Variables:** $Q_k$
  - **Domains:** $\{1, 2, 3, \ldots N\}$
  - **Constraints:**
    - Implicit: $\forall i, j \text{ non-threatening}(Q_i, Q_j)$
    - Explicit: $(Q_1, Q_2) \in \{(1, 3), (1, 4), \ldots\}$
      $\ldots$
Constraint Graphs
Constraint Graphs

- Binary CSP: each constraint relates (at most) two variables

- Binary 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!

[Demo: CSP applet (made available by aispace.org) -- n-queens]
Example: Sudoku

- Variables:
  - Each (open) square
- Domains:
  - \{1,2,\ldots,9\}
- Constraints:
Example: Sudoku

- **Variables:**
  - Each (open) square
- **Domains:**
  - \{1,2,...,9\}
- **Constraints:**

9-way alldiff for each column
Example: Sudoku

- Variables:
  - Each (open) square
- Domains:
  - \{1,2,...,9\}
- Constraints:
  - 9-way alldiff for each column
  - 9-way alldiff for each row
Example: Sudoku

- **Variables:**
  - Each (open) square

- **Domains:**
  - \{1,2,...,9\}

- **Constraints:**
  - 9-way alldiff for each column
  - 9-way alldiff for each row
  - 9-way alldiff for each region
Example: Sudoku

- Variables:
  - Each (open) square
- Domains:
  - \{1,2,...,9\}
- Constraints:
  - 9-way alldiff for each column
  - 9-way alldiff for each row
  - 9-way alldiff for each region
  - (or can have a bunch of pairwise inequality constraints)
Varieties of CSPs and Constraints
Varieties of CSPs

- **Discrete Variables**
  - Finite domains
    - Size $d$ means $O(d^n)$ complete assignments
    - E.g., Boolean CSPs, including Boolean satisfiability (NP-complete)
  - Infinite domains (integers, strings, etc.)
    - E.g., job scheduling, variables are start/end times for each job
    - Linear constraints solvable, nonlinear undecidable
Varieties of CSPs

- **Discrete Variables**
  - Finite domains
    - Size $d$ means $O(d^n)$ complete assignments
    - E.g., Boolean CSPs, including Boolean satisfiability (NP-complete)
  - Infinite domains (integers, strings, etc.)
    - E.g., job scheduling, variables are start/end times for each job
    - Linear constraints solvable, nonlinear undecidable

- **Continuous variables**
  - E.g., start/end times for Hubble Telescope observations
  - Linear constraints solvable in polynomial time by LP methods (see cs170 for a bit of this theory)
Varieties of Constraints

- Varieties of Constraints
  - Unary constraints involve a single variable (equivalent to reducing domains), e.g.:
    
    \[ SA \neq \text{green} \]
  - Binary constraints involve pairs of variables, e.g.:
    
    \[ SA \neq \text{WA} \]
  - Higher-order constraints involve 3 or more variables: e.g., cryptarithmetic column constraints

- Preferences (soft constraints):
  - E.g., red is better than green
  - Often representable by a cost for each variable assignment
  - Gives constrained optimization problems
  - (We’ll ignore these until we get to Bayes’ nets)
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
- Circuit layout
- Fault diagnosis
- ... lots more!

- Many real-world problems involve real-valued variables...
Solving CSPs
Standard Search Formulation

- Standard search formulation of CSPs

- States defined by the values assigned so far (partial assignments)
  - Initial state: the empty assignment, {}
  - Successor function: assign a value to an unassigned variable
  - Goal test: the current assignment is complete and satisfies all constraints

- We’ll start with the straightforward, naïve approach, then improve it
Search Methods

- What would BFS do?
Search Methods

- What would BFS do?

- What would DFS do?

[Demo: coloring -- dfs]
Search Methods

- What would BFS do?
- What would DFS do?
- What problems does naïve search have?

[Demo: coloring -- dfs]
Search Methods

- What would BFS do?
- What would DFS do?
Search Methods

- What would BFS do?
- What would DFS do?
Video of Demo Coloring -- DFS

Search Methods

- What would BFS do?
- What would DFS do?
Backtracking Search
Backtracking Search
Backtracking Search

- Backtracking search is the basic uninformed algorithm for solving CSPs
Backtracking Search

- Backtracking search is the basic uninformed algorithm for solving CSPs

- Idea 1: One variable at a time
  - Variable assignments are commutative, so fix ordering
  - I.e., [WA = red then NT = green] same as [NT = green then WA = red]
  - Only need to consider assignments to a single variable at each step
Backtracking Search

- Backtracking search is the basic uninformed algorithm for solving CSPs

- **Idea 1: One variable at a time**
  - Variable assignments are commutative, so fix ordering
  - I.e., [WA = red then NT = green] same as [NT = green then WA = red]
  - Only need to consider assignments to a single variable at each step

- **Idea 2: Check constraints as you go**
  - I.e. consider only values which do not conflict previous assignments
  - Might have to do some computation to check the constraints
  - “Incremental goal test”
Backtracking search is the basic uninformed algorithm for solving CSPs

- **Idea 1: One variable at a time**
  - Variable assignments are commutative, so fix ordering
  - I.e., [WA = red then NT = green] same as [NT = green then WA = red]
  - Only need to consider assignments to a single variable at each step

- **Idea 2: Check constraints as you go**
  - I.e. consider only values which do not conflict previous assignments
  - Might have to do some computation to check the constraints
  - “Incremental goal test”

- Depth-first search with these two improvements is called *backtracking search* (not the best name)
Backtracking Search

- Backtracking search is the basic uninformed algorithm for solving CSPs.

- **Idea 1: One variable at a time**
  - Variable assignments are commutative, so fix ordering
  - I.e., [WA = red then NT = green] same as [NT = green then WA = red]
  - Only need to consider assignments to a single variable at each step

- **Idea 2: Check constraints as you go**
  - I.e. consider only values which do not conflict previous assignments
  - Might have to do some computation to check the constraints
  - “Incremental goal test”

- Depth-first search with these two improvements is called *backtracking search* (not the best name)

- Can solve n-queens for \( n \approx 25 \)
Backtracking Example
Backtracking Example
Backtracking Example
Backtracking Example
Backtracking Search

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**\((\text{VAR}\,(assignment, csp))\)
    for each value in **ORDER-DOMAIN-VALUES**\((\text{var}, assignment, csp)\) do
        if value is consistent with assignment given \(\text{CONSTRAINTS}[csp]\) then
            add \(\{\text{var} = \text{value}\}\) to assignment
            result ← **RECURSIVE-BACKTRACKING**\((assignment, csp)\)
            if result \(\neq\) failure then return result
        remove \(\{\text{var} = \text{value}\}\) from assignment
    return failure

- Backtracking = DFS + variable-ordering + fail-on-violation

[Demo: coloring -- backtracking]