CS 188: Artificial Intelligence

Constraint Satisfaction Problems II

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

$x_1$ $x_2$
Constraint Satisfaction Problems

$N$ variables
domain $D$
Constraint Satisfaction Problems

N variables
domain D
constraints
Constraint Satisfaction Problems

\( N \text{ variables} \)

\( \text{domain } D \)

\( \text{constraints} \)

\( x_1, x_2 \)

\( \text{states} \)
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
- $x_1$, $x_2$ variables
- Domain $D$
- $x_1$, $x_2$ domain
- Constraints
- States
- Goal test
- Successor function
- Partial assignment
- $C_{map}$
- Map coloring constraints
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
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., \([WA = \text{red} \text{ then NT} = \text{green}]\) same as \([NT = \text{green} \text{ then WA} = \text{red}]\)
  - 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

- 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 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

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

```python
def BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({}, csp)

def 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 \neq\ failure then return result
        remove \{var = value\} from assignment
    return failure
```

[Demo: coloring -- backtracking]
Video of Demo Coloring - Backtracking
Video of Demo Coloring - Backtracking
Video of Demo Coloring - Backtracking
Improving Backtracking

- General-purpose ideas give huge gains in speed
- **Ordering:**
  - Which variable should be assigned next?
  - In what order should its values be tried?
- **Filtering:** Can we detect inevitable failure early?
- **Structure:** Can we exploit the problem structure?
Filtering

Keep track of domains for unassigned variables and cross off bad options
Filtering: Keep track of domains for unassigned variables and cross off bad options.

Forward checking: Cross off values that violate a constraint when added to the existing assignment.

[Demo: coloring -- forward checking]
Filtering: Keep track of domains for unassigned variables and cross off bad options

Forward checking: Cross off values that violate a constraint when added to the existing assignment
Filtering: Forward Checking

- Filtering: Keep track of domains for unassigned variables and cross off bad options.
- Forward checking: Cross off values that violate a constraint when added to the existing assignment.

[Demo: coloring -- forward checking]
Filtering: Keep track of domains for unassigned variables and cross off bad options.

Forward checking: Cross off values that violate a constraint when added to the existing assignment.

[Demo: coloring -- forward checking]
Filtering: Keep track of domains for unassigned variables and cross off bad options.
Forward checking: Cross off values that violate a constraint when added to the existing assignment.

[Demo: coloring -- forward checking]
Filtering: Keep track of domains for unassigned variables and cross off bad options.
Forward checking: Cross off values that violate a constraint when added to the existing assignment.

[Demo: coloring -- forward checking]
Filtering: Keep track of domains for unassigned variables and cross off bad options.

Forward checking: Cross off values that violate a constraint when added to the existing assignment.
Video of Demo Coloring - Backtracking with Forward Checking

Filtering: Forward Checking

- Filtering: Keep track of domains for unassigned variables and cross off bad options
- Forward checking: Cross off values that violate a constraint when added to the existing assignment
Filtering: Forward Checking

- Filtering: Keep track of domains for unassigned variables and cross off bad options
- Forward checking: Cross off values that violate a constraint when added to the existing assignment
Filtering: Forward Checking

- Filtering: Keep track of domains for unassigned variables and cross off bad options
- Forward checking: Cross off values that violate a constraint when added to the existing assignment
Filtering: Constraint Propagation

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

<table>
<thead>
<tr>
<th>WA</th>
<th>NT</th>
<th>Q</th>
<th>NSW</th>
<th>V</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="red" alt="" /></td>
<td><img src="red" alt="" /></td>
<td><img src="green" alt="" /></td>
<td><img src="green" alt="" /></td>
<td><img src="green" alt="" /></td>
<td><img src="red" alt="" /></td>
</tr>
<tr>
<td><img src="red" alt="" /></td>
<td><img src="green" alt="" /></td>
<td><img src="green" alt="" /></td>
<td><img src="green" alt="" /></td>
<td><img src="green" alt="" /></td>
<td><img src="red" alt="" /></td>
</tr>
<tr>
<td><img src="red" alt="" /></td>
<td><img src="green" alt="" /></td>
<td><img src="green" alt="" /></td>
<td><img src="green" alt="" /></td>
<td><img src="green" alt="" /></td>
<td><img src="red" alt="" /></td>
</tr>
</tbody>
</table>
Filtering: 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!
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!
- Why didn’t we detect this yet?
Filtering: 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!
- Why didn’t we detect this yet?
- *Constraint propagation*: reason from constraint to constraint
Consistency of A Single Arc
An arc $X \rightarrow Y$ is **consistent** iff for every $x$ in the tail there is some $y$ in the head which could be assigned without violating a constraint.

- Forward checking: Enforcing consistency of arcs pointing to each new assignment.
Consistency of A Single Arc

- An arc $X \rightarrow Y$ is consistent iff for every $x$ in the tail there is some $y$ in the head which could be assigned without violating a constraint.

- Forward checking: Enforcing consistency of arcs pointing to each new assignment.
Consistency of A Single Arc

- An arc $X \rightarrow Y$ is consistent iff for every $x$ in the tail there is some $y$ in the head which could be assigned without violating a constraint.

- Forward checking: Enforcing consistency of arcs pointing to each new assignment.
Consistency of A Single Arc

- An arc $X \rightarrow Y$ is consistent iff for every $x$ in the tail there is some $y$ in the head which could be assigned without violating a constraint.

- Forward checking: Enforcing consistency of arcs pointing to each new assignment.

![Diagram showing a map of Australia with states WA, NT, Q, NSW, V, SA, and a car with a robot pointing to an assignment.](image)
An arc $X \rightarrow Y$ is consistent iff for every $x$ in the tail there is some $y$ in the head which could be assigned without violating a constraint.

- Forward checking: Enforcing consistency of arcs pointing to each new assignment.

Delete from the tail!
Arc Consistency of an Entire CSP

- A simple form of propagation makes sure all arcs are consistent:

```
WA  NT  Q  NSW  V  SA
```

```
[Diagram showing states and arcs between them]
```

Remember: Delete from the tail!
A simple form of propagation makes sure all arcs are consistent:

Remember: Delete from the tail!
Arc Consistency of an Entire CSP

- A simple form of propagation makes sure all arcs are consistent:

Remember: Delete from the tail!
Arc Consistency of an Entire CSP

- A simple form of propagation makes sure all arcs are consistent:

Remember: Delete from the tail!
Arc Consistency of an Entire CSP

- A simple form of propagation makes sure all arcs are consistent:
A simple form of propagation makes sure all arcs are consistent:

Remember: Delete from the tail!
A simple form of propagation makes sure all arcs are consistent:

- Important: 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
- What’s the downside of enforcing arc consistency?

Remember: Delete from the tail!
Enforcing Arc Consistency in a CSP

- Runtime: \(O(n^2d^3)\), can be reduced to \(O(n^2d^2)\)
- ... but detecting all possible future problems is NP-hard - why?

[Demo: CSP applet (made available by aispaces.org) -- n-queens]
Limitations of Arc Consistency

What went wrong here?

[Demo: coloring -- arc consistency]
[Demo: coloring -- forward checking]
Limitations of Arc Consistency

- After enforcing arc consistency:
  - Can have one solution left
  - Can have multiple solutions left
  - Can have no solutions left (and not know it)

- Arc consistency still runs inside a backtracking search!

What went wrong here?

[Demo: coloring -- arc consistency]
[Demo: coloring -- forward checking]
Video of Demo Coloring - Backtracking with Forward Checking - Complex Graph
Video of Demo Coloring - Backtracking with Forward Checking - Complex Graph
Video of Demo Coloring - Backtracking with Forward Checking - Complex Graph
Video of Demo Coloring - Backtracking with Arc Consistency - Complex Graph
Video of Demo Coloring - Backtracking with Arc Consistency - Complex Graph
Video of Demo Coloring - Backtracking with Arc Consistency - Complex Graph