Computer Science CPSC 322

Lecture 8

Intro to CSP

CSP as Search
Announcements

- Midterm Thursday Feb 16 (see course schedule)
- Posted in Connect ("midterm" folder):
  - Learning goals for the material covered
  - Short questions on material for the whole course.
    - Both midterm and final will have questions very similar, or even verbatim, from this set (about 40-60% of overall mark)
    - You can answer these questions by studying the slides and textbook
      - Cover questions as we proceed through the material, come to see us if you have doubts on how to answer
      - BUT DO STUDY THE RELEVANT MATERIAL FIRST!
    - Do not wait until just before the midterm to review the questions, if you start now it will be much easier to cover all the relevant ones before the exam
- Assignment 1 tomorrow @ 7pm (solution posted 4 days after, to accommodate for late days)
- Assign 2 out in the next couple of days (watch for the announcement). Due Friday Feb 10
Lecture Overview

Recap of previous lecture

- Intro to CSP
- CSP algorithms using Search
  - Generate and test
  - Graph search
- Intro to Arc Consistency (time permitting)
# Recap (Must Know How to Fill This)

<table>
<thead>
<tr>
<th>Selection</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DFS</strong></td>
<td>LIFO</td>
<td>N</td>
<td>N</td>
<td>(O(b^m))</td>
</tr>
<tr>
<td><strong>BFS</strong></td>
<td>FIFO</td>
<td>Y</td>
<td>Y</td>
<td>(O(b^m))</td>
</tr>
<tr>
<td><strong>IDS</strong></td>
<td>LIFO</td>
<td>Y</td>
<td>Y</td>
<td>(O(b^m))</td>
</tr>
<tr>
<td><strong>LCFS</strong></td>
<td>min cost</td>
<td>Y **</td>
<td>Y **</td>
<td>(O(b^m))</td>
</tr>
<tr>
<td><strong>Best First</strong></td>
<td>min h</td>
<td>N</td>
<td>N</td>
<td>(O(b^m))</td>
</tr>
<tr>
<td><strong>A</strong>*</td>
<td>min f</td>
<td>Y**</td>
<td>Y**</td>
<td>(O(b^m))</td>
</tr>
<tr>
<td><strong>B&amp;B</strong></td>
<td>LIFO + pruning</td>
<td>Y**</td>
<td>Y**</td>
<td>(O(b^m))</td>
</tr>
<tr>
<td><strong>IDA</strong>*</td>
<td>LIFO</td>
<td>Y**</td>
<td>Y**</td>
<td>(O(b^m))</td>
</tr>
<tr>
<td><strong>MBA</strong>*</td>
<td>min f</td>
<td>Y**</td>
<td>Y**</td>
<td>(O(b^m))</td>
</tr>
</tbody>
</table>

** Needs conditions: you need to know what they are
# Algorithms Often Used in Practice

<table>
<thead>
<tr>
<th></th>
<th>Selection</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td>LIFO</td>
<td>N</td>
<td>N</td>
<td>$O(b^m)$</td>
<td>$O(mb)$</td>
</tr>
<tr>
<td>BFS</td>
<td>FIFO</td>
<td>Y</td>
<td>Y</td>
<td>$O(b^m)$</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>IDS</td>
<td>LIFO</td>
<td>Y</td>
<td>Y</td>
<td>$O(b^m)$</td>
<td>$O(mb)$</td>
</tr>
<tr>
<td>LCFS</td>
<td>min cost</td>
<td>Y **</td>
<td>Y **</td>
<td>$O(b^m)$</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>Best First</td>
<td>min h</td>
<td>N</td>
<td>N</td>
<td>$O(b^m)$</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>A*</td>
<td>min f</td>
<td>Y**</td>
<td>Y**</td>
<td>$O(b^m)$</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>B&amp;B</td>
<td>LIFO + pruning</td>
<td>Y**</td>
<td>Y**</td>
<td>$O(b^m)$</td>
<td>$O(mb)$</td>
</tr>
<tr>
<td>IDA*</td>
<td>LIFO</td>
<td>Y</td>
<td>Y</td>
<td>$O(b^m)$</td>
<td>$O(mb)$</td>
</tr>
<tr>
<td>MBA*</td>
<td>min f</td>
<td>Y**</td>
<td>Y**</td>
<td>$O(b^m)$</td>
<td>$O(b^m)$</td>
</tr>
</tbody>
</table>

** Needs conditions: you need to know what they are.
Search in Practice

Many paths to solution, no $\infty$ paths?

Informed?

Large branching factor?

These are indeed general guidelines, specific problems might yield different choices
Course Overview

Problem Type
- Static
  - Constraint Satisfaction
  - Query
- Sequential
  - Planning

Deterministic
- Vars + Constraints
- Logics
  - STRIPS
- Search

Stochastic
- Belief Nets
  - Variable Elimination
- Decision Nets
  - Variable Elimination
- Markov Processes
  - Value Iteration

Representation
- Reasoning Technique

Value
- Arc Consistency

First Part of the Course
Standard vs Specialized Search

• We studied general state space search in isolation
  • Standard search problem: search in a state space

• State is a “black box” - any arbitrary data structure that supports three problem-specific routines:
  • goal test: goal(state)
  • finding successor nodes: neighbors(state)
  • if applicable, heuristic evaluation function: h(state)

• We will see more specialized versions of search for various problems
Course Overview

Environment

Deterministic
- Arc Consistency
- Search
- Vars + Constraints

Stochastic
- Belief Nets
  - Variable Elimination
- Decision Nets
  - Variable Elimination
- Markov Processes
  - Value Iteration

Problem Type

Static
- Constraint Satisfaction
  - Query

Sequential
- Planning

Representation
- Reasoning Technique

Logics
- STRIPS
  - Search
We will look at Search in Specific R&R Systems

- Constraint Satisfaction Problems (CPS):
  - State
  - Successor function
  - Goal test
  - Solution
  - Heuristic function

- Query:
  - State
  - Successor function
  - Goal test
  - Solution
  - Heuristic function

- Planning
  - State
  - Successor function
  - Goal test
  - Solution
  - Heuristic function
Course Overview

Problem Type
- Constraint Satisfaction
- Query

Static

Sequential
- Planning

Environment
- Deterministic
  - Arc Consistency
  - Vars + Constraints
  - Search
- Stochastic
  - Belief Nets
    - Variable Elimination
  - Decision Nets
    - Variable Elimination
  - Markov Processes
    - Value Iteration

We’ll start from CPS

Reasoning Technique

Representation

Logics
- STRIPS
  - Search

Value Iteration
We will look at Search for CSP

- Constraint Satisfaction Problems (CPS):
  - State
  - Successor function
  - Goal test
  - Solution
  - Heuristic function

- Query:
  - State
  - Successor function
  - Goal test
  - Solution
  - Heuristic function

- Planning
  - State
  - Successor function
  - Goal test
  - Solution
  - Heuristic function
Lecture Overview

• Recap of previous lecture

Intro to CSP
  • CSP algorithms using Search
    - Generate and test
    - Graph search
  • Intro to Arc Consistency (time permitting)
CSPs: Crossword Puzzles - Proverb

**Daily Puzzles**

- 95.3% words correct (miss three or four words per puzzle)
- 98.1% letters correct
- 46.2% puzzles completely correct

370 puzzles from 7 sources.

**Summary statistics:**

**Proverb Daily Results**

Source: Michael Littman
Constraint Satisfaction Problems (CSP)

- In a CSP
  - state is defined by a set of variables $V_i$ with values from domain $D_i$
  - goal test is a set of constraints specifying
    1. allowable combinations of values for subsets of variables (hard constraints)
    2. preferences over values of variables (soft constraints)
Dimensions of Representational Complexity (from lecture 2)

- Reasoning tasks (Constraint Satisfaction / Logic & Probabilistic Inference / Planning)
- Deterministic versus stochastic domains

Some other important dimensions of complexity:

- **Explicit state** or **features** or **relations**
- Flat or hierarchical representation
- Knowledge given versus knowledge learned from experience
- Goals versus complex preferences
- Single-agent vs. multi-agent
Explicit State vs. Features (Lecture 2)

How do we model the environment?

• You can enumerate the possible states of the world

• A state can be described in terms of features
  • Assignment to (one or more) features
  • Often the more natural description
  • 30 binary features can represent
    $$2^{30} = 1,073,741,824$$ states
Variables/Features and Possible Worlds

- **Variable**: a synonym for feature
  - We denote variables using capital letters
  - Each variable $V$ has a domain $\text{dom}(V)$ of possible values

- **Variables can be of several main kinds:**
  - Boolean: $|\text{dom}(V)| = 2$
  - Finite: $|\text{dom}(V)|$ is finite
  - Infinite but discrete: the domain is countably infinite
  - Continuous: e.g., real numbers between 0 and 1

- **Possible world:**
  - Complete assignment of values to each variable
  - This is equivalent to a state as we have defined it so far

  ✓ Soon, however, we will give a broader definition of state, so it is best to start distinguishing the two concepts.
### Example (lecture 2)

#### Mars Explorer Example

<table>
<thead>
<tr>
<th>Domain</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weather</td>
<td>{S, C}</td>
</tr>
<tr>
<td>Temperature</td>
<td>[-40, 40]</td>
</tr>
<tr>
<td>Longitude</td>
<td>[0, 359]</td>
</tr>
<tr>
<td>Latitude</td>
<td>[0, 179]</td>
</tr>
</tbody>
</table>

One possible world (state) \{S, -30, 320, 210\}

Number of possible (mutually exclusive) worlds (states)

\[
2 \times 81 \times 360 \times 180
\]

Product of cardinality of each domain … always exponential in the number of variables
How many possible worlds?

- Crossword Puzzle 1:
  - variables are words that have to be filled in
  - domains are English words of correct length
  - possible worlds: all ways of assigning words

- Number of English words? Let’s say 150,000
  - Of the right length? Assume for simplicity: 15,000 for each length

- Number of words to be filled in? 63

- How many possible worlds? (assume any combination is ok)
How many possible worlds?

Crossword Puzzle 1:

- variables are words that have to be filled in
- domains are English words of correct length
- possible worlds: all ways of assigning words

- Number of English words? Let’s say 150,000
- Of the right length? Assume for simplicity: 15,000 for each length
- Number of words to be filled in? 63
- How many possible worlds? (assume any combination is ok)

A. $15,000\times63$
B. $15,000^{63}$
C. $63^{15,000}$
D. $1,563^{63}$
How many possible worlds?

• Crossword Puzzle:
  • variables are words that have to be filled in
  • domains are English words of correct length
  • possible worlds: all ways of assigning words

• Number of English words? Let’s say 150,000
  • Of the right length? Assume for simplicity: 15,000 for each word

• Number of words to be filled in? 63

• How many possible worlds? (assume any combination is ok)
  \[15000^{63}\]
How many possible worlds?

• Crossword 2:
  • variables are cells (individual squares)
  • domains are letters of the alphabet
  • possible worlds: all ways of assigning letters to cells

• Number of empty cells? $15 \times 15 - 32 = 193$

• Number of letters in the alphabet? 26

• How many possible worlds? (assume any combination is ok) $26^{193}$

• In general: (domain size) $\#\text{variables}$
Constraint Satisfaction Problems (CSP)

- Allow for usage of useful general-purpose algorithms with more power than standard search algorithms.
- They exploit the multi-dimensional nature of the problem and the structure provided by the goal set of constraints, *not* black box.
Lecture Overview

• Recap of previous lecture

• Intro to CSP
  • Constraints and models

• CSP algorithms using Search
  - Generate and test
  - Graph search

• Intro to Arc Consistency (time permitting)
Constraints

• Constraints are restrictions on the values that one or more variables can take
  
  • Unary constraint: restriction involving a single variable
  • k-ary constraint: restriction involving k different variables
    ✓ We will mostly deal with binary constraints

• Constraints can be specified by

  1. listing all combinations of valid domain values for the variables participating in the constraint
  2. giving a function that returns true when given values for each variable which satisfy the constraint
Constraints: Simple Example

• **Unary constraint:** $V_2 \neq 2$

• **k-ary constraint:**
  - ✓ binary (k=2): $V_1 + V_2 < 5$
  - ✓ 3-ary: $V_1 + V_2 + V_4 < 5$

We will mostly deal with binary constraints

• Constraints can be specified by
  1. listing all combinations of valid domain values for the variables participating in the constraint
     - for constraint $V_1 > V_2$
       and $\text{dom}(V_1) = \{1,2,3\}$ and $\text{dom}(V_2) = \{1,2\}$:

       | $V_1$ | $V_2$ |
       |-------|-------|
       | 2     | 1     |
       | 3     | 1     |
       | 3     | 2     |

  2. giving a function (predicate) that returns true if given values for each variable satisfy the constraint else false:

     $V_1 > V_2$
Examples

- Crossword Puzzle 1:
  - variables are words that have to be filled in
  - domains are valid English words
  - constraints: words have the same letters at points where they intersect

$$h_1[0] = v_1[0]$$
$$h_1[1] = v_2[0]$$

......

~225 constraints
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.,

<table>
<thead>
<tr>
<th>WA</th>
<th>NT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Or

<table>
<thead>
<tr>
<th>NT</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NT</th>
<th>QU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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., $WA \neq NT, NT \neq SA, NT \neq QU, \ldots,$

<table>
<thead>
<tr>
<th>WA</th>
<th>NT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>Green</td>
</tr>
<tr>
<td>Red</td>
<td>Blue</td>
</tr>
<tr>
<td>Green</td>
<td>Red</td>
</tr>
<tr>
<td>Green</td>
<td>Blue</td>
</tr>
<tr>
<td>Blue</td>
<td>Red</td>
</tr>
<tr>
<td>Blue</td>
<td>Green</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NT</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>Green</td>
</tr>
<tr>
<td>Red</td>
<td>Blue</td>
</tr>
<tr>
<td>Green</td>
<td>Red</td>
</tr>
<tr>
<td>Green</td>
<td>Blue</td>
</tr>
<tr>
<td>Blue</td>
<td>Red</td>
</tr>
<tr>
<td>Blue</td>
<td>Green</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NT</th>
<th>QU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>Green</td>
</tr>
<tr>
<td>Red</td>
<td>Blue</td>
</tr>
<tr>
<td>Green</td>
<td>Red</td>
</tr>
<tr>
<td>Green</td>
<td>Blue</td>
</tr>
<tr>
<td>Blue</td>
<td>Red</td>
</tr>
<tr>
<td>Blue</td>
<td>Green</td>
</tr>
</tbody>
</table>

..............
Example: Eight Queen problem

- Eight Queen problem: place 8 queens on a chessboard so that no queen can attack the others

- Constraints: No queens can be in the same row, column or diagonals
Example 2: 8 queens

- **Variables:** \( V_1, \ldots, V_n \)
  
  \( V_i = \text{Row occupied by the } i^{th} \text{ queen in the } i^{th} \text{ column} \)

- **Domains:** \( D_{V_i} = \{1,2,3,4,5,6,7,8\} \)

- **Constraints:** Two queens cannot be on the same row, column or on the same diagonal
Example: Eight Queen problem
Example 2: 8 queens

• **Variables:** $V_1, \ldots, V_n$
  
  $V_i = \text{Row occupied by the } i^{\text{th}} \text{ queen in the } i^{\text{th}} \text{ column}$

• **Domains:** $D_{V_i} = \{1,2,3,4,5,6,7,8\}$

• **Constraints:** Two queens cannot be on the same row, column or on the same diagonal

• We can specify the constraints by enumerating explicitly, for each pair of columns, which positions are allowed. Ex:
  
  Constr($V_1$, $V_2$) =
Example 2: 8 queens

• Variables: \( V_1, \ldots, V_n \)
  \( V_i = \) Row occupied by the \( i^{th} \) queen in the \( i^{th} \) column

• Domains: 
  \( D_{V_i} = \{1,2,3,4,5,6,7,8\} \)

• Constraints: 
  Two queens cannot be on the same row or on the same diagonal

• We can specify the constraints by enumerating explicitly, for each pair of columns, what positions are allowed. Ex:
  
\[
\text{Constr}(V_1, V_2) = \{(1,3), (1,4), \ldots (1,8) \}
\]
\[
(2,4)(2,5)\ldots
\]
\[
(8,1)\ldots (8,5), (8,6)\}
\]
Constraints: one more concept

- Constraints are restrictions on the values that one or more variables can take
  - Unary constraint: restriction involving a single variable
  - k-ary constraint: restriction involving k different variables
    - ✓ We will mostly deal with binary constraints
  - Constraints can be specified by
    1. listing all combinations of valid domain values for the variables participating in the constraint
    2. giving a function that returns true when given values for each variable which satisfy the constraint

- A possible world satisfies a set of constraints
  - if the values for the variables involved in each constraint are consistent with that constraint, i.e.
    1. They are elements of the list of valid domain values
    2. Function for that constraint returns true for those values
Constraint Satisfaction Problems (CSPs): Definitions

Definition:
A constraint satisfaction problem (CSP) consists of:
  • a set of variables V
  • a domain \( \text{dom}(V) \) for each variable
  • a set of constraints C

Definition:
A model of a CSP is an assignment of values to all of its variables (i.e., a possible world) that satisfies all of its constraints.

Simple example:
  • \( V = \{V_1\} \)
    - \( \text{dom}(V_1) = \{1,2,3,4\} \)
  • \( C = \{C_1,C_2\} \)
    - \( C_1: V_1 \neq 2 \)
    - \( C_2: V_1 > 1 \)

All models for this CSP:
**Constraint Satisfaction Problems (CSPs): Definitions**

**Definition:**
A constraint satisfaction problem (CSP) consists of:
- a set of variables $V$
- a domain $\text{dom}(V)$ for each variable
- a set of constraints $C$

**Definition:**
A model of a CSP is an assignment of values to all of its variables (i.e., a possible world) that satisfies all of its constraints.

**Simple example:**
- $V = \{V_1\}$
  - $\text{dom}(V_1) = \{1,2,3,4\}$
- $C = \{C_1,C_2\}$
  - $C_1: V_1 \neq 2$
  - $C_2: V_1 > 1$

**All models for this CSP:**
- $V_1 = 3$
- $V_1 = 4$
Models and Possible Worlds

Definition:
A model of a CSP is an assignment of values to all of its variables (i.e. a possible world) that satisfies all of its constraints.

Another example:
- \( V = \{V_1, V_2\} \)
  - \( \text{dom}(V_1) = \{1,2,3\} \)
  - \( \text{dom}(V_2) = \{1,2\} \)
- \( C = \{C_1, C_2, C_3\} \)
  - \( C_1: V_2 \neq 2 \)
  - \( C_2: V_1 + V_2 < 5 \)
  - \( C_3: V_1 > V_2 \)

How many models do we have?

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

i-clicker.
Models and Possible Worlds

Definition:
A **model** of a CSP is an assignment of values to all of its variables (i.e. a possible world) that satisfies all of its constraints.

\[ V = \{V_1, V_2\} \]
- \( \text{dom}(V_1) = \{1,2,3\} \)
- \( \text{dom}(V_2) = \{1,2\} \)

\[ C = \{C_1, C_2, C_3\} \]
- \( C_1: V_2 \neq 2 \)
- \( C_2: V_1 + V_2 < 5 \)
- \( C_3: V_1 > V_2 \)

Possible worlds for this CSP:
- \{\(V_1=1, V_2=1\}\}
- \{\(V_1=1, V_2=2\}\}
- \{\(V_1=2, V_2=1\}\) (a model)
- \{\(V_1=2, V_2=2\}\}
- \{\(V_1=3, V_2=1\}\) (a model)
- \{\(V_1=3, V_2=2\}\}

\[ \text{end} \]
Definition:
A **model** of a CSP is an assignment of values to all of its variables (i.e. **a possible world**) that satisfies all of its constraints.

WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green
Scope of a constraint

Definition:
The scope of a constraint is the set of variables that are involved in the constraint.

- Examples:
  - $V_2 \neq 2$ has scope $\{V_2\}$
  - $V_1 > V_2$ has scope $\{V_1, V_2\}$
  - $V_1 + V_2 + V_4 < 5$ has scope $\{V_1, V_2, V_4\}$

- Number of variables in the scope of a k-ary constraint ...
Finite Constraint Satisfaction Problem: Definition

Definition:
A finite constraint satisfaction problem (FCSP) is a CSP with a finite set of variables and a finite domain for each variable.

• We will only study finite CSPs here
  ✓ but many of the techniques carry over to countably infinite and continuous domains. We use CSP here to refer to FCSP.

• In FCSP, the scope of each constraint is automatically finite since it is a subset of the finite set of variables.
Constraint Satisfaction Problems: Variants

• We may want to solve the following problems with a CSP:
  • determine whether or not a model exists
  • find a model
  • find all of the models
  • count the number of models
  • find the best model, given some measure of model quality
    ✓ this is now an optimization problem
  • determine whether some property of the variables holds in all models
Solving Constraint Satisfaction Problems

• Even the simplest problem of determining whether or not a model exists in a general CSP with finite domains is NP-hard
  • There is no known algorithm with worst case polynomial runtime.
  • We can't hope to find an algorithm that is polynomial for all CSPs.

• However, we can try to:
  • find efficient (polynomial) consistency algorithms that reduce the size of the search space
  • identify special cases for which algorithms are efficient
  • work on approximation algorithms that can find good solutions quickly, even though they may offer no theoretical guarantees
  • find algorithms that are fast on typical (not worst case) cases
Lecture Overview

• Recap of previous lecture

• Intro to CSP
  • Constraints and models

  CSP algorithms using Search
    - Generate and test
    - Graph search

• Intro to Arc Consistency (time permitting)
Generate and Test (GT) Algorithms

- Systematically check all possible worlds
  - Possible worlds: cross product of domains
    \[ \text{dom}(V_1) \times \text{dom}(V_2) \times \cdots \times \text{dom}(V_n) \]

- Generate and Test:
  - Generate possible worlds one at a time
  - Test constraints for each one.

Example: 3 variables A, B, C

```python
For a in dom(A)
    For b in dom(B)
        For c in dom(C)
            if \{A=a, B=b, C=c\} satisfies all constraints
                return \{A=a, B=b, C=c\}
            fail
```
Generate and Test (GT) Algorithms

For a in dom(A)
    For b in dom(B)
        For c in dom(C)
            if {A=a, B=b, C=c} satisfies all constraints
                return {A=a, B=b, C=c}
        fail

If there are $k$ variables, each with domain size $d$, and there are $c$ constraints, the complexity of Generate & Test

A. $O(ckd)$
B. $O(ck^d)$
C. $O(cd^k)$
D. $O(d^{ck})$
Generate and Test (GT) Algorithms

- If there are $k$ variables, each with domain size $d$, and there are $c$ constraints, the complexity of Generate & Test is

$$O(cd^k)$$

- There are $d^k$ possible worlds
- For each one need to check $c$ constraints
Generate and Test (GT) Algorithms

- Need to find a better way, that exploits the “insights” that we have on the goal expressed in terms of constraints
  - Why does GT fail to do this?
Generate and Test (GT) Algorithms

- Need to find a better way, that exploits the “insights” that we have on the goal expressed in terms of constraints
  - Why does GT fail to do this?

- It checks contraints only after a full assignment of values to variables has been made
  - Fails to leverage the modular/explicit nature of the goal
CSP as a Search Problem: one formulation

• States: partial assignments of values to variables
• Start state: empty assignment
• Successor function: states with the next variable assigned
  • Follow a total order of the variables \( V_1, \ldots, V_n \)
  • A state assigns values to the first \( k \) variables:
    \[ \{ V_1 = v_1, \ldots, V_k = v_k \} \]
  • Neighbors of node \( \{ V_1 = v_1, \ldots, V_k = v_k \} \):
    nodes \( \{ V_1 = v_1, \ldots, V_k = v_k, V_{k+1} = x \} \) for each \( x \in \text{dom}(V_{k+1}) \)

• Goal states: complete assignments of values to variables that satisfy all constraints
  • That is, models
• Solution: assignment (the path does not matter)
CSP as a Search Problem: one formulation

\[
\begin{align*}
&V_1 = v_1 \\
&V_2 = v_1 \\
&V_3 = v_2 \\
&V_1 = v_1 \\
&V_2 = v_1 \\
&V_3 = v_2 \\
&V_1 = v_1 \\
&V_2 = v_2 \\
&V_1 = v_1 \\
&V_2 = v_k \\
&V_1 = v_k
\end{align*}
\]
Which search algorithm would be most appropriate for this formulation of CSP?

A. Depth First Search
B. BFS
C. A *
D. IDS
Which search algorithm would be most appropriate for this formulation of CSP?

A. Depth First Search

- the search tree is always finite and has no cycles
- If there are n variables every solution is at depth n.
- Possibly very very large branching factor b
Dealing with complexity

• CSP problems can be huge
  - Thousands of variables
    • Exponentially more search states
  - Exhaustive search is typically infeasible

• Many algorithms exploit the structure provided by the goal ⇒ set of constraints, *not* black box
Backtracking algorithms

• Explore search space via DFS but evaluate each constraint as soon as all its variables are bound.

• Any partial assignment that does not satisfy the constraint can be pruned.

• Example:
  - 3 variables A, B, C each with domain \{1,2,3,4\}
  - \{A = 1, B = 1\} is inconsistent with constraint \(A \neq B\) regardless of the value of the other variables
    \[
    \Rightarrow \text{Fail! Prune!}
    \]
CSP as Graph Searching

Check unary constraints on $V_1$
If not satisfied = PRUNE

Check constraints on $V_1$ and $V_2$ If not satisfied = PRUNE
Solving CSPs by DFS: Example

- Variables: A, B, C
- Domains: \{1, 2, 3, 4\}
- Constraints: A < B, B < C

Good ordering, lots of pruning happens right away
Solving CSPs by DFS: Example Efficiency

- Variables: A, B, C
- Domains: \{1, 2, 3, 4\}
- Constraints: A < B, B < C

Problem? Performance heavily depends on the order in which variables are considered.

Much worse ordering, keeps more branches around.
EXAMPLE: $D_A = D_B = D_C = \{1,2,3,4\}$

$\neg (A = B) \land \neg (B = C) \land (C < D) \land (A = D) \land (B \neq D) \land (E < A) \land (E < B) \land (E < C) \land (E < D)$

A=1 B=1 failure

B=2 C=1 D=1 failure

D=2 failure

D=3 failure

D=4 failure

C=2 failure

C=3 D=1 failure

D=2 failure

D=3 failure

D=4 failure

C=4 D=1 failure

D=2 failure

D=3 failure

D=4 failure

\ldots
EXAMPLE: $D_A = D_B = D_C = \{1, 2, 3, 4\}$

$$(A \neq B) \land (B \neq C) \land (C < D) \land (A = D) \land (B \neq D) \land (E < A) \land (E < B) \land (E < C) \land (E < D)$$

A=1 D=1 C= 1 failure
C=2 failure
C=3 failure
C=4 failure
D=2 failure
D=3 failure
D=4 failure

A=1 B=1 failure

B=2 C=1 D=1 failure
D=2 failure
D=3 failure
D=4 failure

C=2 failure
C=3 D=1 failure
D=2 failure
D=3 failure
D=4 failure

C=4 D=1 failure
D=2 failure
D=3 failure
D=4 failure
Selecting variables in a smart way

- Backtracking relies on one or more heuristics to select which variables to consider next.
  - E.g, variable involved in the largest number of constraints:
  - Can also be smart about which values to consider first

- But we will look at an alternative approach that can do much better
Meaning of ‘heuristic’ in CSP as search

• This is a different use of the word ‘heuristic’ from the definition we have given in the search module
  - Qualifications of ‘heuristic’ still true in the context of CSP as search
    • Can be computed cheaply during the search
    • Provides guidance to the search algorithm
  - Qualification that is not true anymore in this context
    • ‘Estimate of the distance to the goal’

• Both meanings are used frequently in the AI literature.
• In general
  • ‘heuristic’ means ‘serves to discover’: goal-oriented.
  • Does not mean ‘unreliable’!
Standard Search vs. Specific R&R systems

- Constraint Satisfaction (Problems):
  - State: assignments of values to a subset of the variables
  - Successor function: assign values to a "free" variable
  - Goal test: all variables assigned a value and all constraints satisfied?
  - Solution: possible world that satisfies the constraints
  - Heuristic function: none (all solutions at the same distance from start)

- Planning:
  - State
  - Successor function
  - Goal test
  - Solution
  - Heuristic function

- Inference:
  - State
  - Successor function
  - Goal test
  - Solution
  - Heuristic function
Learning Goals for CSP

• Define possible worlds in term of variables and their domains
• Compute number of possible worlds on real examples
• Specify constraints to represent real world problems differentiating between:
  • Unary and k-ary constraints
  • List vs. function format
• Verify whether a possible world satisfies a set of constraints (i.e., whether it is a model, a solution)
• Implement the Generate-and-Test Algorithm. Explain its disadvantages.
• Solve a CSP by search (specify neighbors, states, start state, goal state).
• Compare strategies for CSP search.
• Implement pruning for DFS search in a CSP.
Lecture Overview

• Recap of previous lecture

• Intro to CSP
  • Constraints and models

• CSP algorithms using Search
  - Generate and test
  - Graph search

Intro to Arc Consistency (time permitting)
Can we do better than Search?

Key idea

• prune the domains as much as possible before searching for a solution.

**Definition:** A variable is domain consistent if no value of its domain is ruled impossible by any unary constraints.

• Example: \( \text{dom}(V) = \{1, 2, 3, 4\} \).

• Variable \( V \) is not domain consistent with the constraint \( V \neq 2 \)

  - It is domain consistent once we remove 2 from its domain.

**Pruning domains** is trivial for unary constraints. Trickier for k-ary ones.
Def. A constraint network is defined by a graph, with
- one node for every variable (drawn as circle)
- one node for every constraint (drawn as rectangle)
- undirected edges running between variable nodes and constraint nodes whenever a given variable is involved in a given constraint.

- Three variables: A, B, C
- Two constraints: A<B, A>C
Def. A constraint network is defined by a graph, with
- one node for every variable (drawn as circle)
- one node for every constraint (drawn as rectangle)
- undirected edges running between variable nodes and constraint nodes whenever a given variable is involved in a given constraint.

Example:

- Variables: A, B, C
- Domains: \{1, 2, 3, 4\}
- 3 Constraints: A < B, B < C, B = 3
Def. A constraint network is defined by a graph, with
- one node for every variable (drawn as circle)
- one node for every constraint (drawn as rectangle)
- undirected edges running between variable nodes and constraint nodes whenever a given variable is involved in a given constraint.

Example:

• Variables: A, B, C
• Domains: {1, 2, 3, 4}
• 3 Constraints: A < B, B < C, B = 3

5 edges/arcs in the constraint network:
- \langle A, A < B \rangle, \langle B, A < B \rangle
- \langle B, B < C \rangle, \langle C, B < C \rangle
- \langle B, B = 3 \rangle
A more complicated example

How many variables are there in this constraint network?

How many constraints
A more complicated example

How many variables are there in this constraint network?

A. 5  B. 6  C. 9  D. 14
A more complicated example

How many variables are there in this constraint network?

A. 5  B. 6  C. 9  D. 14
A more complicated example

How many variables are there in this constraint network?

A. 5
B. 9
C. 14
D. 18

How many constraints?

A. 6
B. 9
C. 14
D. 18
A more complicated example

How many variables are there in this constraint network?

A. 5
B. 9
C. 14
D. 18

How many constraints?

A. 6
B. 9
C. 14
D. 18
Definition:
An arc \( <x, r(x,y)> \) is **arc consistent** if for each value \( x \) in \( \text{dom}(X) \) there is some value \( y \) in \( \text{dom}(Y) \) such that \( r(x,y) \) is satisfied.

**A network is arc consistent** if all its arcs are arc consistent.
Arc Consistency

Definition:
An arc \(<x, r(x,y)>\) is **arc consistent** if for each value \(x\) in \(\text{dom}(X)\) there is some value \(y\) in \(\text{dom}(Y)\) such that \(r(x,y)\) is satisfied.

A network is **arc consistent** if all its arcs are arc consistent.

---

Not arc consistent:
No value in domain of B that satisfies \(A < B\) if \(A = 3\)

Arc consistent: Both \(B = 2\) and \(B = 3\) have ok values for A
- e.g. for \(B = 2\) \(A = 1\)