Lecture 4b: Constraint Satisfaction Problems (CSPs)
CS 580 (001) - Spring 2016

Amarda Shehu

Department of Computer Science
George Mason University, Fairfax, VA, USA

February 17, 2016
1 Outline of Today’s Class

2 CSP Examples

3 Backtracking Search for CSPs

4 Problem Structure and Problem Decomposition

5 Local Search for CSPs

6 Take-home Problem

7 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

e.g., \(WA \neq NT\) (if the language allows this), or

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

Solutions are assignments satisfying all constraints, e.g.,
\[
\{ \text{WA} = \text{red}, \text{NT} = \text{green}, \text{Q} = \text{red}, \text{NSW} = \text{green}, \text{V} = \text{red}, \text{SA} = \text{blue}, \text{T} = \text{green} \}
\]
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 $d \Rightarrow O(d^n)$ 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., $StartJob_1 + 5 \leq StartJob_3$
◊ 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., $SA \neq \text{green}$

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

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

Strong vs. soft constraints

Preferences (soft constraints)
e.g., red is better than green
often representable by a cost for each variable assignment
$\rightarrow$ 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

\( \text{alldiff}(F, T, U, W, R, O) \)
\( O + O = R + 10 \cdot X_1, \text{ etc.} \)
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
Standard Search Formulation (Incremental)

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.  
  $\implies$ 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  
  $\implies$ 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.,
\[
[WA = red \text{ then } NT = green] \quad \text{same as} \quad [NT = green \text{ then } WA = red]
\]

Only need to consider assignments to a single variable at each node
\[\implies b = d \text{ and there are } d^n \text{ leaves}\]

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

Amarda Shehu (580)
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
**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
**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
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* 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 \( y \)

![Diagram of Arc Consistency]

WA  NT  Q  NSW  V  SA  T
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
\( X \rightarrow Y \) 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
Arc Consistency Algorithm

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 Remove-Inconsistent-Values(\(X_i, X_j\)) then
    for each \(X_k\) in Neighbors[\(X_i\)] do
      add \((X_k, X_i)\) to queue

function Remove-Inconsistent-Values(\(X_i, X_j\)) returns true iff succeeds

\(removed \leftarrow false\)

for each \(x\) in Domain[\(X_i\)] do
  if no value \(y\) in Domain[\(X_j\)] allows \((x,y)\) to satisfy the constraint \(X_i \leftrightarrow X_j\)
  then delete \(x\) from Domain[\(X_i\)]
  \(removed \leftarrow true\)

return \(removed\)
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 $\text{deg}(X_i)$
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?

\[ \sum \text{over all degrees } = O(c) \]

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 \((X_k, X_i)\) 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

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
   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?
   \( \text{sum over all degrees is } O(E) \) of constraint graph
   which is \( O(c) \)

How often will the domain of each variable be pruned?

\( O(d^2) \) times
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 \( \deg(X_i) \)

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
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
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?
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?
   $O(cd)$
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 $(X_k, X_i)$ 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?
  $O(cd)$

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 $(X_k, X_i)$ 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: $T(\text{AC}−3) \in O(cd^3)$
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
   ...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 \( \frac{n}{c} \cdot d^c \), linear in \( n \).

E.g., \( n = 80, \ d = 2, \ c = 20 \)

\[
2^{80} = 4 \text{ billion years at 10 million nodes/sec}
\]

\[
4 \cdot 2^{20} = 0.4 \text{ seconds at 10 million nodes/sec}
\]
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 $\text{RemoveInconsistent}(\text{Parent}(X_j), X_j)$.

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

---

Problem Structure and Problem Decomposition
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

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 \texttt{reassign} variable values

Variable selection: randomly select any conflicted variable

Value selection by \texttt{min-conflicts} heuristic:
- choose value that violates the fewest constraints
- i.e., hill-climber with $h(n) = \text{total number of violated constraints}$

Take-home: Propose a simple EA for 4-queens CSP
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
**States:** 4 queens in 4 columns ($4^4 = 256$ states)
**Example: 4-Queens as CSP**

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

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

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

**Operators:** move queen in column
Example: 4-Queens as CSP

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

**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 \( (4^4 = 256 \text{ states}) \)

**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:** $h(n) = \text{number of attacks}$
**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:** \(h(n) = \text{number of attacks}\)
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** \( Q_1, Q_2, Q_3, Q_4 \)
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**

\[
Q_i \neq Q_j \quad \text{(cannot be in same row)}
\]

\[
|Q_i - Q_j| 
eq |i - j| \quad \text{(or same diagonal)}
\]
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**
\[
Q_i \neq Q_j \text{ (cannot be in same row)} \\
|Q_i - Q_j| \neq |i - j| \text{ (or same diagonal)}
\]

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 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