AI Search

Chao Lan
Outline

Introduction

Uninformed Search vs. Informed Search

Constraint Satisfaction Problem

Adversarial Search
CSP and Three Solvers

- Constraint Propagation
- Backtracking Search
- Local Search
What is CSP?

When each state is described by multiple variables, assigning all variables with values satisfying preset constraints is the constraint satisfaction problem (CSP).
Example CSP: Map Coloring
Example CSP: Class Scheduling
Example CSP: N-Queens Problem
So, why do we want to formalize problems as CSP?
Now, let us be more formal...
A CSP has three components

- a set of **variables** $X = \{X_1, \ldots, X_n\}$
- a set of **domains** $D = \{D_1, \ldots, D_n\}$
- a set of **constraints** $C = \{C_1, \ldots, C_k\}$, e.g., $C_1: X_1 \neq X_2$, $C_2: X_5 < X_3$
Example

Map coloring problem has

$X = \_\_\_\_$

$D = \_\_\_\_$

$C = \_\_\_\_$
Example

Map coloring problem has

\[ X = \{ (X_1) \text{NW}, (X_2) \text{HE}, \ldots \} \]

\[ D = \]

\[ C = \]
Example

Map coloring problem has

\[ X = \{ (X_1)\text{NW}, (X_2)\text{HE}, \ldots \} \]

\[ D = \{ (D_1)\{R,G,B,Y\}, (D_2)\{R,G,B,Y\}, \ldots \} \]

\[ C = \]
Example

Map coloring problem has

$X = \{ (X_1)\text{NW}, (X_2)\text{HE}, \ldots \}$

$D = \{ (D_1)\{\text{R,G,B,Y}\}, (D_2)\{\text{R,G,B,Y}\}, \ldots \}$

$C = \{ (C_1)\text{NW} \neq \text{HE}, (C_2)\text{NW} \neq \text{NI}, \ldots \}$
A state is an assignment of values to some or all variables.
- e.g., State 1 = \{ NW=Y, NE=R, \ldots \}
A state is an assignment of values to some or all variables.
- e.g., State 1 = { NW=Y, NE=R, … }

A **consistent assignment** is one without constraint violation.
A state is an assignment of values to some or all variables.
- e.g., State 1 = \{ NW=Y, NE=R, \ldots \}

A **consistent assignment** is one without constraint violation.
- Q: is \{ TH=Y, NI=R, ST=Y, \ldots \} a consistent assignment?
A state is an assignment of values to some or all variables.
- e.g., State 1 = \{ NW=Y, NE=R, \ldots \} 

A consistent assignment is one without constraint violation.
- Q: is \{ TH=Y, NI=R, ST=Y, \ldots \} a consistent assignment?

A **complete assignment** is one with all variables assigned.
A state is an assignment of values to some or all variables.
- e.g., State 1 = \{ NW=Y, NE=R, \ldots \} 

A consistent assignment is one without constraint violation.
- Q: is \{ TH=Y, NI=R, ST=Y, \ldots \} a consistent assignment?

A **complete assignment** is one with all variables assigned.
- Q: what would be a **partial assignment**?
A state is an assignment of values to some or all variables.
- e.g., State 1 = \{ NW=Y, NE=R, \ldots \} 

A consistent assignment is one without constraint violation.
- Q: is \{ TH=Y, NI=R, ST=Y, \ldots \} a consistent assignment?

A complete assignment is one with all variables assigned.
- Q: what would be a partial assignment?

A **solution** is a complete, consistent assignment.
We can visualize CSP as a constraint graph...
Visualize CSP as a Constraint Graph
And there are many types of CSP...
CSP Categorization

Domain: finite vs. infinite

Domain: discrete vs. continuous

Constraint: linear vs. non-linear

Constraint: unary vs. binary vs. global
Three Constraint Types

Unary constraint restricts value of a single variable.
- e.g., WA ≠ Red, NSW = Blue or Green

Binary constraint relates to variables.
- e.g., WA ≠ SA, Q ≠ NSW

Global constraint involves an arbitrary number of variables.
Constraint propagation reduces legal values for a variable using constraints.
Constraint propagation reduces legal values for a variable using constraints.

- reduce search space
- can intertwine with search or as preprocessor
- use different types of local consistency
  - node consistency
  - arc consistency
  - path consistency
  - k-consistency
Node Consistency for C.P.

A variable is node-consistent if all values in its domain satisfy unary constraints.
A variable is node-consistent if all values in its domain satisfy unary constraints.

- e.g., if SA dislikes green, then $D_{SA} = \{\text{red, blue, yellow}\}$ makes SA consistent
A variable is node-consistent if all values in its domain satisfy unary constraints.

- e.g., if SA dislikes green, then $D_{SA} = \{\text{red, blue, yellow}\}$ makes SA consistent.

A network is node-consistent if every variable is node-consistent.
Node Consistency for C.P.

A variable is node-consistent if all values in its domain satisfy unary constraints.
- e.g., if SA dislikes green, then $D_{SA} = \{\text{red, blue, yellow}\}$ makes SA consistent

A network is node-consistent if every variable is node-consistent.
- one can often eliminate unary constraints by running node consistency (preprocess)

We will assume unary constraints are eliminated in the following discussions.
Constraint propagation reduces legal values for a variable using constraints.

- reduce search space
- can intertwine with search or as preprocessor
- use different types of local consistency
  - node consistency
  - arc consistency
  - path consistency
  - k-consistency
A variable is *arc-consistent* if every value in its domain satisfy binary constraints.
A variable is arc-consistent if every value in its domain satisfy binary constraints.

Example: SA is arc-consistent with respect to NT, if for every $x \in D(SA)$, there is a $y \in D(NT)$ such that $(x,y)$ satisfies binary constraint on arc (SA, NT), e.g., $SA \neq NT$. 
Arc Consistency for C.P.

A variable is arc-consistent if every value in its domain satisfy binary constraints.

Example: SA is arc-consistent with respect to NT, if for every $x \in D(SA)$, there is a $y \in D(NT)$ such that $(x,y)$ satisfies binary constraint on arc (SA, NT), e.g., SA\(\neq\)NT.

Q: is SA arc-consistent w.r.t. NT if…?

1. $D(SA) = \{\text{red, blue}\}$, $D(NT) = \{\text{green}\}$
A variable is arc-consistent if every value in its domain satisfy binary constraints.

Example: SA is arc-consistent with respect to NT, if for every \( x \in D(SA) \), there is an \( y \in D(NT) \) such that \((x,y)\) satisfies binary constraint on arc \((SA, NT)\), e.g., \( SA \neq NT \).

Q: is SA arc-consistent w.r.t. NT if…?

1. \( D(SA) = \{\text{red, blue}\}, D(NT) = \{\text{green}\} \)
2. \( D(SA) = \{\text{red, blue, green}\}, D(NT) = \{\text{green}\} \)
A variable is arc-consistent if every value in its domain satisfy binary constraints.

Example: SA is arc-consistent with respect to NT, if for every $x \in D(SA)$, there is a $y \in D(NT)$ such that $(x,y)$ satisfies binary constraint on arc $(SA, NT)$, e.g., $SA \neq NT$.

Q: is SA arc-consistent w.r.t. NT if…?

1. $D(SA) = \{\text{red, blue}\}$, $D(NT) = \{\text{green}\}$
2. $D(SA) = \{\text{red, blue, green}\}$, $D(NT) = \{\text{green}\}$
3. $D(SA) = \{\text{red, blue, green}\}$, $D(NT) = \{\text{green, red}\}$
Achieve Arc Consistency by AC-3 Algorithm

function AC-3(csp) returns false if an inconsistency is found and true otherwise
inputs: csp, a binary CSP with components (X, D, C)
local variables: queue, a queue of arcs, initially all the arcs in csp

while queue is not empty do
    (X_i, X_j) ← REMOVE-FIRST(queue)
    if REVISE(csp, X_i, X_j) then
        if size of D_i = 0 then return false
        for each X_k in X_i.NEIGHBORS - {X_j} do
            add (X_k, X_i) to queue
    return true

function REVISE(csp, X_i, X_j) returns true iff we revise the domain of X_i
revised ← false
for each x in D_i do
    if no value y in D_j allows (x, y) to satisfy the constraint between X_i and X_j then
        delete x from D_i
        revised ← true
return revised
Basic Idea of AC-3 Algorithm

Check every arc \((X, Y)\) to see if \(X\) is arc-consistent w.r.t. \(Y\).

- if yes, leave arc \((X, Y)\)

- if not, modify \(D(X)\) to achieve arc consistent, and recheck \((Z, X)\) for every \(Z\)

Repeat until every arc satisfies arc-consistent.
Basic Idea of AC-3 Algorithm

Check every arc $(X, Y)$ to see if $X$ is arc-consistent w.r.t. $Y$.

- if yes, leave arc $(X, Y)$
- if not, modify $D(X)$ to achieve arc consistent, and recheck $(Z, X)$ for every $Z$

Repeat until every arc satisfies arc-consistent.

The actual algorithm uses a queue to store all arcs that need to be checked.
Basic Idea of AC-3 Algorithm

Check every arc \((X, Y)\) to see if \(X\) is arc-consistent w.r.t. \(Y\).

- if yes, leave arc \((X, Y)\)

- if not, modify \(D(X)\) to achieve arc consistent, and recheck \((Z, X)\) for every \(Z\)

Repeat until every arc satisfies arc-consistent.

Q: why recheck \((Z, X)\) for every \(Z\) if \(D(X)\) is modified?
Complexity of AC-3 Arc Checking

Given $n$ variables, each having domain size $\leq d$ and constraint number $\leq c$. 
Given $n$ variables, each having domain size $\leq d$ and constraint number $\leq c$.

Q: for one variable & one constraint, how many arcs we need to check?
Complexity of AC-3 Arc Checking

Given n variables, each having domain size \( \leq d \) and constraint number \( \leq c \).

Q: for one variable & one constraint, how many arcs we need to check? \( O(d^2) \)

Q: for one variable, how many arcs we need to check?
Complexity of AC-3 Arc Checking

Given $n$ variables, each having domain size $\leq d$ and constraint number $\leq c$.

Q: for one variable & one constraint, how many arcs we need to check? $O(d^2)$

Q: for one variable, how many arcs we need to check? $O(c \times d^2)$

Q: for the network, how many arcs we need to check?
Complexity of AC-3 Arc Checking

Given n variables, each having domain size ≤ d and constraint number ≤ c.

Q: for one variable & one constraint, how many arcs we need to check? O(d^2)

Q: for one variable, how many arcs we need to check? O(c*d^2)

Q: for the network, how many arcs we need to check? O(n*c*d^2)

Note multiplicative constants can be absorbed by the big O notation.
Solve CSP by Constraint Propagation

Constraint propagation reduces legal values for a variable using constraints.

- reduce search space
- can intertwine with search or as preprocessor
- use different types of local consistency
  - node consistency
  - arc consistency
  - path consistency
  - k-consistency
Path Consistency for C.P.

“Path consistency tightens the binary constraints by using implicit constraints that are inferred by looking at triples of variables.”

We say two nodes (SA, NSW) is path-consistent with respect to a third node Q, if for every consistent \((x,y) \in (D(SA),D(NSW))\) there is a \(z \in D(Q)\) such that \((x, z)\) and \((y, z)\) satisfy binary constraints on arc \((SA, Q)\) and arc \((NSW, Q)\), respectively, e.g., \(SA \neq Q, NSW \neq Q\).
Path Consistency for C.P.

“Path consistency tightens the binary constraints by using implicit constraints that are inferred by looking at triples of variables.”

We say two nodes (SA, NSW) is path-consistent with respect to a third node Q, if for every consistent \((x, y) \in (D(SA), D(NSW))\) there is a \(z \in D(Q)\) such that \((x, z)\) and \((y, z)\) satisfy binary constraints on arc \((SA, Q)\) and arc \((NSW, Q)\), respectively, e.g., \(SA\neq Q, NSW\neq Q\).

Q: is \((NT, Q)\) path-consistent w.r.t. SA if…?

1. \(D(NT) = \{blue\}, \quad D(Q) = \{red\}, \quad D(SA) = \{green\}\)
2. \(D(NT) = \{blue, red\}, \quad D(Q) = \{red\}, \quad D(SA) = \{green\}\)
3. \(D(NT) = \{blue\}, \quad D(Q) = \{red\}, \quad D(SA) = \{blue\}\)
4. \(D(NT) = \{blue, red\}, \quad D(Q) = \{red, green\}, \quad D(SA) = \{green\}\)
Path Consistency for C.P.

“Path consistency tightens the binary constraints by using implicit constraints that are inferred by looking at triples of variables.”

We say two nodes (SA, NSW) is path-consistent with respect to a third node Q, if for every consistent (x,y) ∈ (D(SA),D(NSW)) there is a z ∈ D(Q) such that (x, z) and (y, z) satisfy binary constraints on arcs (SA, Q) and (NSW, Q), respectively, e.g., SA≠Q, NSW≠Q.

Q: is (NT,Q) path-consistent w.r.t. SA if…?
1. D(NT)={blue}, D(Q)={red}, D(SA) = {green}
2. D(NT)={blue, red}, D(Q)={red}, D(SA) = {green}
3. D(NT)={blue}, D(Q)={red}, D(SA) = {blue}
4. D(NT)={blue, red}, D(Q)={red, green}, D(SA) = {green}

Path-consistency can be achieved by PC-2 algorithm, which is similar to AC-3 algorithm.
Constraint propagation reduces legal values for a variable using constraints.

- reduce search space
- can intertwine with search or as preprocessor
- use different types of local consistency
  - node consistency
  - arc consistency
  - path consistency
  - k-consistency
**K-Consistency for C.P.**

*K-consistency* is a generalization of path-consistency to multiple variables.

A CSP is k-consistent if, for every k-1 variables $X_1, X_2, \ldots, X_{k-1}$ with any consistent assignment, there is always an assignment to any other variable $X_k$, such that $(X_1, X_k), (X_2, X_k), \ldots, (X_{k-1}, X_k)$ are arc-consistent, e.g., $X_j \neq X_k$ for all $j=1,\ldots,k-1$. 
K-Consistency is a generalization of path-consistency to multiple variables.

A CSP is k-consistent if, for every k-1 variables $X_1, X_2, \ldots, X_{k-1}$ with any consistent assignment, there is always an assignment to any other variable $X_k$, such that $(X_1, X_k), (X_2, X_k), \ldots, (X_{k-1}, X_k)$ are arc-consistent, e.g., $X_j \neq X_k$ for all $j=1, \ldots, k-1$.

Q: is the example problem 3-consistent if...

1. domain of all variables = \{red, blue, yellow, green\}
K-Consistency is a generalization of path-consistency to multiple variables.

A CSP is k-consistent if, for every k-1 variables $X_1, X_2, \ldots, X_{k-1}$ with any consistent assignment, there is always an assignment to any other variable $X_k$, such that $(X_1, X_k), (X_2, X_k), \ldots, (X_{k-1}, X_k)$ are arc-consistent, e.g., $X_j \neq X_k$ for all $j=1, \ldots, k-1$.

Q: is the example problem 3-consistent if...

1. domain of all variables = \{red, blue, yellow, green\}

2. domain of all variables = \{red, blue, yellow\}
CSP and Three Solvers

- Constraint Propagation
- Backtracking Search
- Local Search
What’s wrong with standard search in CSP?

Consider depth-first search.

Each state is a partial assignment.

Each action is to assign one variable.
What’s wrong with standard search in CSP?

Consider depth-first search.

Each state is a partial assignment.

Each action is to assign one variable.

Q: how many nodes will be generated?

domain size = d, variable size = n
What’s wrong with standard search in CSP?

Consider depth-first search.

Each state is a partial assignment.

Each action is to assign one variable.

Q: how many nodes will be generated?
A: \( nd \times (n-1) d \times (n-2) d \times \ldots = O(n! \times d^n) \)

\* Very inefficient, as there are only \( d^n \) possible assignments!
Commutativity

A problem is **commutative** if the order of applying any set of actions has no effect on the outcome.
Commutativity

A problem is commutative if the order of applying any set of actions has no effect on the outcome.

All CSPs are commutative. Thus we can examine one variable at each step. Complexity is down to $O(d^n)$.

Q: how is complexity down to $O(d^n)$ now?

domain size = d, variable size = n
Backtracking Search

Backtracking Search = depth-first + one variable per time + backtrack

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

BS saves memory by not actually generating all successors, but only one and keeps changing its values.
Three Ways to Improve Backtracking

Q1: which variable to assign next?

Q2: which value to assign next?

Q3: perform inference?
Three Ways to Improve Backtracking

Q1: which variable to assign next?

Q2: which value to assign next?

Q3: perform inference?
Minimum-Remaining-Values (MRV) Heuristic

MRV first assigns the variable with the fewest legal values.
Minimum-Remaining-Values (MRV) Heuristic

MRV first assigns the variable with the fewest legal values.

- Q: after WA is assigned, which region will be assigned next by MRV?
Minimum-Remaining-Values (MRV) Heuristic

MRV first assigns the variable with the fewest legal values.

- Q: after WA is assigned, which region will be assigned next by MRV?
- Q: after WA & NT is assigned, which region will be assigned next by MRV?

domain = {red, green, blue}
Minimum-Remaining-Values (MRV) Heuristic

MRV first assigns the variable with the fewest legal values.

- Q: after WA is assigned, which region will be assigned next by MRV?
- Q: after WA & NT is assigned, which region will be assigned next by MRV?

Q: what is the rationale behind MRV?
Minimum-Remaining-Values (MRV) Heuristic

MRV first assigns the variable with the fewest legal values.

- Q: after WA is assigned, which region will be assigned next by MRV?
- Q: after WA & NT is assigned, which region will be assigned next by MRV?

Q: what is the rationale behind MRV?
A: shrink search space more quickly

Q: how to choose the first node?
A: no solution… we can use degree heuristic

domain = {red, green, blue}
Degree heuristic first assigns the variable which is involved in the largest number of constraints on other unassigned variables.
Degree Heuristic

Degree heuristic first assigns the variable which is involved in the largest number of constraints on other unassigned variables.

- Q: Which region will degree heuristic start at root node?
Degree Heuristic

Degree heuristic first assigns the variable which is involved in the largest number of constraints on other unassigned variables.

- Q: which region will degree heuristic start at root node?
- Q: after (SA, NT) are assigned, D.H. will assign Q or NSW. (true or false?)
Degree Heuristic

Degree heuristic first assigns the variable which is involved in the largest number of constraints on other unassigned variables.

- Q: which region will degree heuristic start at root node?
- Q: after (SA, NT) are assigned, D.H. will assign Q or NSW. (true or false?)

Q: what’s the rationale behind degree?
Degree Heuristic

Degree heuristic first assigns the variable which is involved in the largest number of constraints on other unassigned variables.

- Q: which region will degree heuristic start at root node?
- Q: after (SA, NT) are assigned, D.H. will assign Q or NSW. (true or false?)

Q: what’s the rationale behind degree?

Degree heuristic is often less powerful than MRV, but can be used as tier-breaker.
Three Ways to Improve Backtracking

Q1: which variable to assign next?

Q2: which value to assign next?

Q3: perform inference?
Least-Constraining-Value (LCV) Heuristic

LCV first assigns the value that rules out fewest choices for neighboring variables.
Least-Constraining-Value (LCV) Heuristic

LCV first assigns the value that rules out fewest choices for neighboring variables.

- Q: if WA = red, NT = green, what value will LCV assign to Q?

domain = {red, green, blue}
Least-Constraining-Value (LCV) Heuristic

LCV first assigns the value that rules out fewest choices for neighboring variables.

- Q: if WA = red, NT = green, what value will LCV assign to Q?

Q: what is the rationale behind LCV?
Least-Constraining-Value (LCV) Heuristic

LCV first assigns the value that rules out fewest choices for neighboring variables.

- Q: if WA = red, NT = green, what value will LCV assign to Q?

Q: what is the rationale behind LCV?

A: want to increase flexibility to find one solution quickly.
Least-Constraining-Value (LCV) Heuristic

LCV first assigns the value that rules out fewest choices for neighboring variables.

- Q: if WA = red, NT = green, what value will LCV assign to Q?

Q: what is the rationale behind LCV?
A: want to increase flexibility to find one solution quickly.
R: if we want to find all solutions, LCV does not help.

domain = \{red, green, blue\}
Three Ways to Improve Backtracking

Q1: which variable to assign next?

Q2: which value to assign next?

Q3: perform inference?

Every time a variable is assigned, we can re-run inference to shrink search space before proceeding to the next variable.
Forward Checking Inference

FC establishes arc consistency between assigned and unassigned variables.

Specifically, if $X$ is assigned $X = x$, and $Y$ is connected to $X$ by a constraint, FC will delete any value $y \in D(Y)$ that are not arc-consistent with $x$. 
Forward Checking Inference

FC establishes arc consistency between assigned and unassigned variables.

Specifically, if X is assigned X = x, and Y is connected to X by a constraint, FC will delete any value \( y \in D(Y) \) that are not arc-consistent with x.

EX: after WA=red, remove “red” from D(NT) before assigning NT.
An Example of Feedforward Checking

<table>
<thead>
<tr>
<th></th>
<th>WA</th>
<th>NT</th>
<th>Q</th>
<th>NSW</th>
<th>V</th>
<th>SA</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial domains</td>
<td>R</td>
<td>G</td>
<td>B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
</tr>
<tr>
<td>After WA=red</td>
<td>🟥</td>
<td></td>
<td></td>
<td>R</td>
<td></td>
<td>R G B</td>
<td>R G B</td>
</tr>
<tr>
<td>After Q=green</td>
<td>🟥</td>
<td></td>
<td>🟢</td>
<td>R</td>
<td></td>
<td>R G B</td>
<td>B</td>
</tr>
<tr>
<td>After V=blue</td>
<td>🟥</td>
<td></td>
<td>🟢</td>
<td>R</td>
<td></td>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

- WA
- NT
- Q
- NSW
- V
- SA
- T

Map:

- Western Australia
- Northern Territory
- Queensland
- South Australia
- New South Wales
- Victoria
- Tasmania

Diagram:

- WA
- NT
- Q
- NSW
- V
- SA
- T
We can combine variable selection (MRV) with inference (feedforward).
Example: MRV + Forward Checking

Limitation

Feedforward only checks consistency between current variable and its neighbors, but does not look ahead on other variables. Maintaining Arc Consistency (MAC) addresses this problem.
CSP and Three Solvers

- Constraint Propagation
- Backtracking Search
- Local Search
Local Search

Local search starts with a complete variable assignment, and continually modify variables that violate some constraints, one at a time, until a solution is found.
Local Search

Local search starts with a complete variable assignment, and continually modify variables that violate some constraints, one at a time, until a solution is found.

N-Queens Problem
1. random assignment per col
2. modify col-8 queen
3. modify col-6 queen
4. continue until converge
Local Search

Local search starts with a complete variable assignment, and continually modifies variables that violate some constraints, one at a time, until a solution is found.

N-Queens Problem

1. random assignment per col
2. modify col-8 queen
3. modify col-6 queen
4. continue until converge

We can assign multiple new values to an inconsistent variable. Q: which new value to try first?
Min-Conflict Heuristic for Local Search

MC first assigns the value that results in fewest violations of the modified variable.
Advantages of Local Search

It is efficient, solving million-queens problem in an average of 50 steps.

It can incorporate weight of constraints - constraint weighting.

It can be applied in an online setting when problem changes.

- backtracking with new constraints require more time
- backtracking with new constraints can result in very different solution
We can improve search efficiency by exploiting the *structure* of the problem.
Example

Q: does the coloring of Tasmania affect the coloring of other regions?
Exploit Structure by Divide-and-Conquer

Idea: divide the original problem into easier subproblems, solve them separately, and combine their solutions to form the final solution.
Exploit Structure by Divide-and-Conquer

Idea: divide the original problem into easier subproblems, solve them separately, and combine their solutions to form the final solution.

Case 1: subproblems are independent

Case 2: subproblems are trees
Exploit Structure by Divide-and-Conquer

Idea: divide the original problem into easier subproblems, solve them separately, and combine their solutions to form the final solution.

Case 1: subproblems are independent

Case 2: subproblems are trees
Independent Subproblems

Ideally, we hope to obtain independent subproblems

- can be ascertained by finding connected components in the constraint graph

If solution to subproblem i is Si, then $U_i S_i$ is solution to the original problem.

We can easily prove search is way more efficient.

A connected component is a subgraph whose nodes are connected to each other by paths but not to any other node.
Search Efficiency over Subproblems

Original problem has \( n \) variables, each having domain size \( d \).

Q: what is search cost?
Search Efficiency over Subproblems

Original problem has $n$ variables, each having domain size $d$.

Q: what is search cost? $O(d^n)$, which is *exponential* in $n$. 
Search Efficiency over Subproblems

Original problem has n variables, each having domain size d.
Q: what is search cost? $O(d^n)$, which is exponential in n.

Divide original problem in subproblems, each having c variables.
Q: what is search cost of a subproblem?
Search Efficiency over Subproblems

Original problem has n variables, each having domain size d.

Q: what is search cost? $O(d^n)$, which is exponential in n.

Divide original problem in subproblems, each having c variables.

Q: what is search cost of a subproblem? $O(d^c)$. 
Search Efficiency over Subproblems

Original problem has \( n \) variables, each having domain size \( d \).

Q: what is search cost? \( \mathcal{O}(d^n) \), which is exponential in \( n \).

Divide original problem in subproblems, each having \( c \) variables.

Q: what is search cost of a subproblem? \( \mathcal{O}(d^c) \).

Q: what is total search cost?
Search Efficiency over Subproblems

Original problem has n variables, each having domain size d.
Q: what is search cost? $O(d^n)$, which is exponential in n.

Divide original problem in subproblems, each having c variables.
Q: what is search cost of a subproblem? $O(d^c)$.

Q: what is total search cost? $O(n \cdot d^c)$, which is linear in n.
Why is $O(n^d^c)$ so great?
Exploit Structure by Divide-and-Conquer

Idea: divide the original problem into easier subproblems, solve them separately, and combine their solutions to form the final solution.

Case 1: subproblems are independent

Case 2: subproblems are trees
When subproblems are trees.

A constraint graph is a tree if any two variables are connected by only one path.
When subproblems are trees.

A constraint graph is a tree if any two variables are connected by *only* one path.

Q: which of the following graphs is a tree?
When subproblems are trees.

A constraint graph is a tree if any two variables are connected by only one path.

Q: which of the following graphs is a tree?

Claim: any tree-structured CSP can be solved in time linear in the number of variables.
Three-Step Proof

1. get a **topological sort** of the tree
   - pick any variable to be the root of the tree, and choose an ordering of the variables such that each variable appears after its parent in the tree
Three-Step Proof

2. run arc-consistency on the sort

Backward Algorithm: removes inconsistent (Parent(Xi), Xi) backwards
Three-Step Proof

2. run arc-consistency on the sort

Backward Algorithm: removes inconsistent (Parent(Xi), Xi) backwards
Three-Step Proof

2. run arc-consistency on the sort

Claim 1: after backward, all arcs are consistent

Claim 2: complexity is $O(n \cdot d^2)$ for $n$ nodes

Backward Algorithm: removes inconsistent (Parent(Xi), Xi) backwards
Three-Step Proof

3. find a solution for the sort using forward assignment
   - assign values forward
   - should be easy and efficient because the sort is arc-consistent
   - may return contradiction if there is no consistent values to choose
Algorithm of Solving Tree in CSP

\[
\text{function } \text{TREE-CSP-SOLVER}(csp) \text{ returns a solution, or failure}
\]
\[
\text{inputs: } csp, \text{ a CSP with components } X, D, C
\]
\[
n \leftarrow \text{number of variables in } X
\]
\[
\text{assignment} \leftarrow \text{an empty assignment}
\]
\[
\text{root} \leftarrow \text{any variable in } X
\]
\[
X \leftarrow \text{TOPOLOGICALSORT}(X, \text{root})
\]
\[
\text{for } j = n \text{ down to } 2 \text{ do}
\]
\[
\text{MAKE-ARC-CONSISTENT(PARENT}(X_j), X_j)
\]
\[
\text{if it cannot be made consistent then return failure}
\]
\[
\text{for } i = 1 \text{ to } n \text{ do}
\]
\[
\text{assignment}[X_i] \leftarrow \text{any consistent value from } D_i
\]
\[
\text{if there is no consistent value then return failure}
\]
\[
\text{return assignment}
\]
Today...

The basic idea is to divide the original problem into easier subproblems, solve them separately, and then combine their solutions to form the final solution.

Case 1: subproblems are independent

Case 2: subproblems are trees

Q: how to reduce a problem into trees?
- cutset
- decomposition
Today...

The basic idea is to divide the original problem into easier subproblems, solve them separately, and then combine their solutions to form the final solution.

Case 1: subproblems are independent

Case 2: subproblems are trees

Q: how to reduce a problem into trees?
  - cutset
  - decomposition
Generate Tree by Cutset Method

Obs: a constraint graph may become a tree if we remove some nodes from it.
- the removed set of nodes is called a cycle cutset
Generate Tree by Cutset Method

Implement: remove a node by fixing its value
Generate Tree by Cutset Method

Implement: remove a node by fixing its value, and run arc-consistency on it.
Algorithm

1. choose a cycle cutset $S$
   A c.c. is a set of variables one can remove to convert a constraint graph into trees.

2. for each possible assignment of $S$, solve the remaining CSP.
   The remaining CSP is a tree pruned to be consistent with $S$.

3. combine solution of the new CSP (if any) and assigned $S$. 
Exercise: find a cutset
The basic idea is to divide the original problem into easier subproblems, solve them separately, and then combine their solutions to form the final solution.

Case 1: subproblems are independent

Case 2: subproblems are trees

Q: how to reduce a problem into trees?
- cutset
- decomposition
Generate Tree by Tree Decomposition

Idea: decompose the original problem into (connected) subproblems, solve them independently, and combine their results to find a solution.
Generate Tree by Tree Decomposition

Idea: decompose the original problem into (connected) subproblems, solve them independently, and combine their results to find a solution.

- subproblems need not be independent

- take additional effort to combine results of subproblems to find a solution

- if one subproblem has no solution, the entire problem has no solution
Three Rules of Tree Decomposition

1. every variable in the original problem appears in at least one subproblem.

2. if two variables are connected by a constraint in the original problem, they must appear together (along with the constraint) in at least one subproblem.

3. if a variable appears in two subproblems in the tree, it must appear in every subproblem along the path connecting those subproblems. (value consistency)
Exercise

What’s wrong with the tree decomposition?

1. every variable in the original problem appears in at least one subproblem.

2. if two variables are connected by a constraint in the original problem, they must appear together (with the constraint) in at least one subproblem.

3. if a variable appears in two subproblems in the tree, it must appear in every subproblem along the path connecting those subproblems.
Combine Results of Subproblems

Each subproblem may generate multiple solutions. We can choose one solution that is consistent across subproblems, e.g.,

SP1.Sol1: \{WA=red, SA=blue, NT=green\}
SP1.Sol2: \{WA=blue, SA=red, NT=green\}
SP2.Sol1: \{Q=red, SA=blue, NT=green\}
SP2.Sol2: \{Q=blue, SA=green, NT=red\}

Finding a solution consistent across subproblems is also a CSP with meta-variables.
Complexity based on Tree Decomposition

Tree width of a tree decomposition of a graph is one less than the size of the largest subproblem.

Tree width of a graph is the minimum tree width among all its tree decompositions.

If a graph has tree width $w$, and we have the corresponding tree decomposition, the problem is solvable in $O(n*d^{(w+1)})$. 
Complexity based on Tree Decomposition

Tree width of a tree decomposition of a graph is one less than the size of the largest subproblem, e.g., 2.

Tree width of a graph is the minimum tree width among all its tree decompositions.

If a graph has tree width $w$, and we have the corresponding tree decomposition, the problem is solvable in $O(n \cdot d^{w+1})$.

Claim: CSPs with constraint graphs of bounded tree width are solvable in polynomial time.
Summary

Backtracking Search (search each node at a time)
- speed up Backtracking by
  - prioritize node to assign (MRV, Degree)
  - prioritize value to assign (LCV)

Local Search (modify inconsistent node)
- prioritize using min-conflict heuristic

Speed up CSP search by working with subproblems
- independent vs tree
- topological sort (complexity linear in problem size)
- generate tree by cutset or decomposition