



CMU 15-381

CSPs

TEACHERS:

DREW BAGNELL

EMMA BRUNSKILL (THIS TIME)

**WITH THANKS TO ARIEL
PROCACCIA AND OTHER PRIOR
INSTRUCTIONS FOR SLIDES**

CLASS SCHEDULING WOES

- 4 more required classes to graduate
 - A: Algorithms
 - B: Bayesian Learning
 - C: Computer Programming
 - D: Distributed Computing
- A few restrictions
 - Algorithms must be taken same semester as distributed computing
 - Computer programming is a prereq for distributed computing and Bayesian learning, so it must be taken in an earlier semester
 - Advanced algorithms and Bayesian Learning are always offered at the same time, so they cannot be taken the same semester
- 3 semesters (semester 1,2,3) when can take classes



CONSTRAINT SATISFACTION PROBLEMS (CSPs)

- *Variables*: $V = \{V_1, \dots, V_N\}$
- *Domain*: Set of d possible values for each variable
- *Constraints*: $C = \{C_1, \dots, C_K\}$
- A constraint consists of
 - variable tuple
 - list of possible values for tuple (ex. $[(V_2, V_3), \{(R, B), (R, G)\}]$)
 - Or function that describes possible values (ex. $V_2 \neq V_3$)
- Allows useful general-purpose algorithms with more power than standard search algorithms



OVERVIEW

- Real world CSPs
- Basic algorithms for solving CSPs
- Pruning space through propagating information



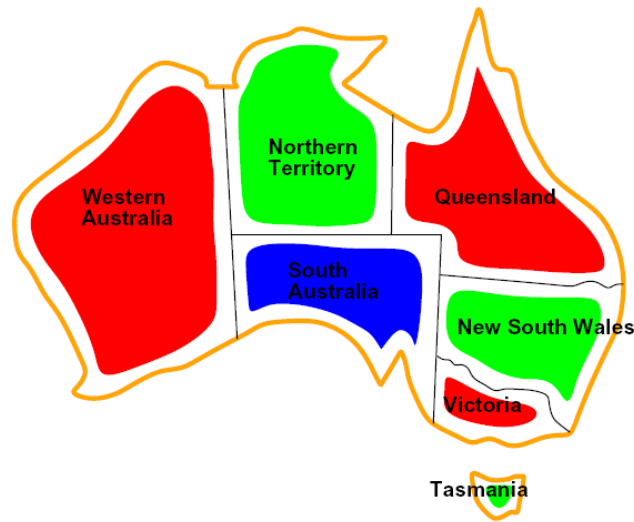
OVERVIEW

- **Real world CSPs**
- Basic algorithms for solving CSPs
- Pruning space through propagating information



EXAMPLE: MAP COLORING

Color a map so that adjacent areas are different colors



MAP COLORING: MATCH!

Constraints

$\{red, green, blue\}$

Variables

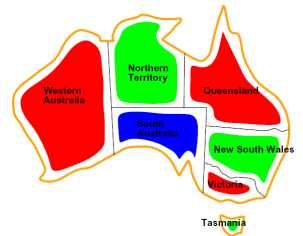
$\{WA = red, NT = green, Q = red,$
 $NSW = green, V = red, SA = blue,$
 $T = green\}$

Domain

WA, NT, Q, NSW

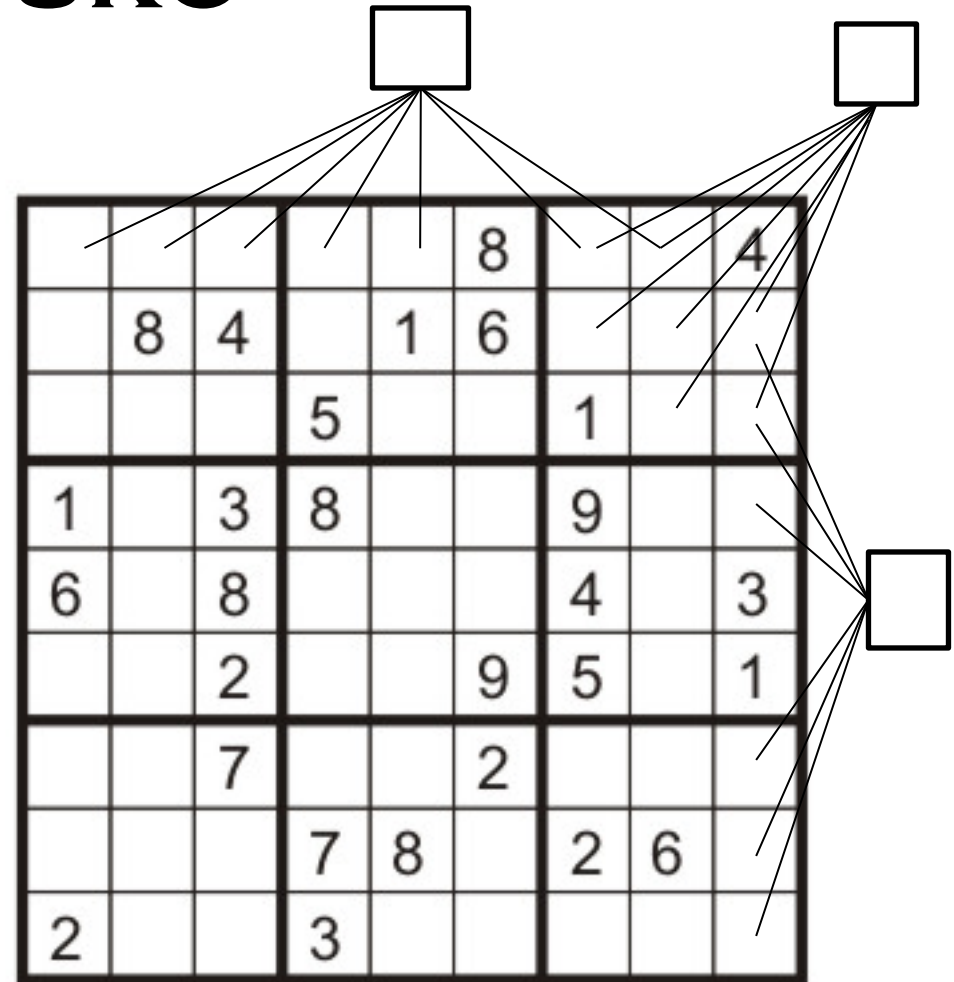
Solutions

$(WA, NT) \in \{(red, green), (red, blue), (green, red), \dots\}$



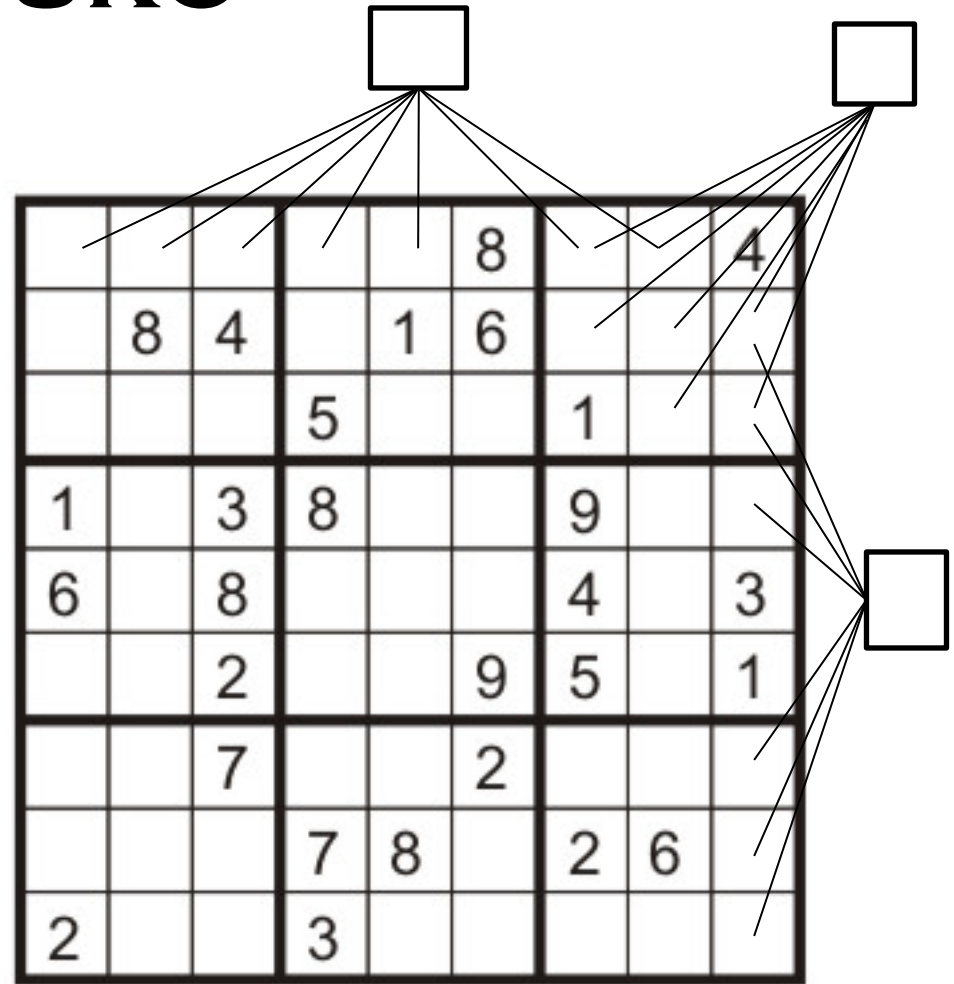
EXAMPLE: SUDUKO

- Variables:
- Domain:
- Constraints:



EXAMPLE: SUDUKO

- Variables:
 - Each open sq
- Domain:
 - {1:9}
- Constraints:
 - 9-way all diff col
 - 9-way all diff row
 - 9-way all diff box



SCHEDULING (IMPORTANT EX.)

- Many industries. Many multi-million \$ decisions. Used extensively for space mission planning. Military uses.
- People *really care* about improving scheduling algorithms! Problems with phenomenally huge state spaces. But for which solutions are needed very quickly
- Many kinds of scheduling problems e.g.:
 - *Job shop*: Discrete time; weird ordering of operations possible; set of separate jobs.
 - *Batch shop*: Discrete or continuous time; restricted operation of ordering; grouping is important.



JOB SCHEDULING

- A set of N jobs, J_1, \dots, J_n .
- Each job j is a seq of operations $O_1^j, \dots, O_{L_j}^j$
- Each operation may use resource R , and has a specific duration in time.
- A resource must be used by a single operation at a time.
- All jobs must be completed by a due time.
- Problem: assign a start time to each job.



EXERCISE: DEFINE CSP

- 4 more required classes to graduate: A, B, C, D
- A must be taken same semester as D
- C is a prereq for D and B so must take C earlier than D & B
- A & B are always offered at the same time, so they cannot be taken the same semester
- 3 semesters (semester 1,2,3) when can take classes



EXERCISE: DEFINE CSP

- 4 more required classes to graduate: A, B, C, D
- A must be taken same semester as D
- C is a prereq for D and B so must take C earlier than D & B
- A & B are always offered at the same time, so they cannot be taken the same semester
- 3 semesters (semester 1,2,3) when can take classes
- Variables: A,B,C,D
- Domain: {1,2,3}
- Constraints: $A \neq B$, $A=D$, $C < B$, $C < D$



OVERVIEW

- Real world CSPs
- **Basic algorithms for solving CSPs**
- Pruning space through propagating information



WHY NOT JUST DO BASIC SEARCH ALGORITHMS FROM LAST TIME?



BACKTRACKING

- Only consider a single variable at each point
- Don't care about path!



BACKTRACKING

- Only consider a single variable at each point
- Don't care about path!
- Order of variable assignment doesn't matter, so fix ordering



BACKTRACKING

- Only consider a single variable at each point
- Don't care about path!
- Order of variable assignment doesn't matter, so fix ordering
- Only consider values which do not conflict with assignment made so far



BACKTRACKING

- Only consider a single variable at each point
- Don't care about path!
- Order of variable assignment doesn't matter, so fix ordering
- Only consider values which do not conflict with assignment made so far
- Depth-first search for CSPs with these two improvements is called backtracking search



BACKTRACKING

- Function `Backtracking(csp)` returns soln or fail
 - Return `Backtrack({},csp)`
- Function `Backtrack(assignment,csp)` returns soln or fail
 - If assignment is complete, return assignment
 - $V_i \leftarrow \text{select_unassigned_var}(csp)$
 - For each `val` in `order-domain-values(var,csp,assign)`
 - If value is consistent with assignment
 - Add $[V_i = \text{val}]$ to assignment
 - $\text{Result} \leftarrow \text{Backtrack}(\text{assignment},csp)$
 - If $\text{Result} \neq \text{fail}$, return result
 - Remove $[V_i = \text{val}]$ from assignments
 - Return fail



BACKTRACKING EXAMPLE

- Variables: A,B,C,D
- Domain: {1,2,3}
- Constraints:
 - $A \neq B$, $A=D$, $C < B$, $C < D$
- Variable order: ?
- Value order: ?



BACKTRACKING

- Function `Backtracking(csp)` returns soln or fail
 - Return `Backtrack({},csp)`
- Function `Backtrack(assignment,csp)` returns soln or fail
 - If assignment is complete, return assignment
 - $V_i \leftarrow \text{select_unassigned_var}(csp)$
 - For each val in ***order-domain-values(var,csp,assign)***
 - If value is consistent with assignment
 - Add $[V_i = \text{val}]$ to assignment
 - $\text{Result} \leftarrow \text{Backtrack}(\text{assignment},csp)$
 - If $\text{Result} \neq \text{fail}$, return result
 - Remove $[V_i = \text{val}]$ from assignments
 - Return fail



CLICK!

- Does variable order affect how long backtracking takes to find a solution?
- **A) Yes**
- B) No



CLICK!

- Does variable order affect the solution found by backtracking?
- **A) Yes**
- B) No



EXAMPLE

VARIABLES: A,B,C,D DOMAIN: {1,2,3}

CONSTRAINTS: $A \neq B$, $A=D$, $C < B$, $C < D$

VARIABLE ORDER: ALPHABETICAL VALUE ORDER: DESCENDING

- (A=3)



EXAMPLE

VARIABLES: A,B,C,D DOMAIN: {1,2,3}

CONSTRAINTS: $A \neq B$, $A=D$, $C < B$, $C < D$

VARIABLE ORDER: ALPHABETICAL VALUE ORDER: **DESCENDING**

- (A=3)
- (A=3, B=3) inconsistent with $A \neq B$
- (A=3, B=2)
- (A=3, B=2, C=3) inconsistent with $C < B$
- (A=3, B=2, C=2) inconsistent with $C < B$
- (A=3, B=2, C=1)
- (A=3, B=2, C=1, D=3) VALID



EXAMPLE

VARIABLES: A,B,C,D DOMAIN: {1,2,3}

CONSTRAINTS: $A \neq B$, $A=D$, $C < B$, $C < D$

VARIABLE ORDER: ALPHABETICAL VALUE ORDER: **ASCENDING**

- (A=1)



EXAMPLE

VARIABLES: A,B,C,D DOMAIN: {1,2,3}

CONSTRAINTS: $A \neq B$, $A=D$, $C < B$, $C < D$

VARIABLE ORDER: ALPHABETICAL VALUE ORDER: **ASCENDING**

- (A=1)
- (A=1,B=1) inconsistent with $A \neq B$
- (A=1,B=2)
- (A=1,B=2,C=1)
- (A=1,B=2,C=1,D=1) inconsistent with $C < D$
- (A=1,B=2,C=1,D=2) inconsistent with $A=D$
- (A=1,B=2,C=1,D=3) inconsistent with $A=D$




EXAMPLE

VARIABLES: A,B,C,D DOMAIN: {1,2,3}

CONSTRAINTS: $A \neq B$, $A=D$, $C < B$, $C < D$

VARIABLE ORDER: ALPHABETICAL

VALUE ORDER: **ASCENDING**

- (A=1)
 - (A=1,B=1) inconsistent with $A \neq B$
 - (A=1,B=2)
 - (A=1,B=2,C=1)
 - (A=1,B=2,C=1,D=1) inconsistent with $C < D$
 - (A=1,B=2,C=1,D=2) inconsistent with $A=D$
 - (A=1,B=2,C=1,D=3) inconsistent with $A=D$
 - No valid assignment for D, return result = fail
 - Backtrack to (A=1,B=2,C=)
 - Try (A=1,B=2,C=2) but inconsistent with $C < B$
 - Try (A=1,B=2,C=3) but inconsistent with $C < B$
 - No other assignments for C, return result= fail
 - Backtrack to (A=1,B=)
 - (A=1,B=3)
 - (A=1,B=3,C=1)
 - (A=1,B=3,C=1,D=1) inconsistent with $C < D$
 - (A=1,B=3,C=1,D=2) inconsistent with $A = D$
 - (A=1,B=3,C=1,D=3) inconsistent with $A = D$
 - Return result = fail
 - Backtrack to (A=1,B=3,C=)
- 
- (A=1,B=3,C=2) inconsistent with $C < B$
 - (A=1,B=3,C=3) inconsistent with $C < B$
 - No remaining assignments for C, return fail
 - Backtrack to (A=1,B=)
 - No remaining assignments for B, return fail
 - Backtrack to A
 - (A=2)
 - (A=2,B=1)
 - (A=2,B=1,C=1) inconsistent with $C < B$
 - (A=2,B=1,C=2) inconsistent with $C < B$
 - (A=2,B=1,C=3) inconsistent with $C < B$
 - No remaining assignments for C, return fail
 - Backtrack to (A=2,B=?)
 - (A=2,B=2) inconsistent with $A \neq B$
 - (A=2,B=3)
 - (A=2,B=3,C=1)
 - (A=2,B=3,C=1,D=1) inconsistent with $C < D$
 - (A=2,B=3,C=1,D=2) **ALL VALID**
-

ORDERING MATTERS!

- Function `Backtracking(csp)` returns soln or fail
 - Return `Backtrack({},csp)`
- Function `Backtrack(assignment,csp)` returns soln or fail
 - If assignment is complete, return assignment
 - $V_i \leftarrow \text{select_unassigned_var}(csp)$
 - For each val in ***order-domain-values(var,csp,assign)***
 - If value is consistent with assignment
 - Add $[V_i = \text{val}]$ to assignment
 - Result $\leftarrow \text{Backtrack}(assignment,csp)$
 - If Result \neq fail, return result
 - Remove $[V_i = \text{val}]$ from assignments
 - Return fail



MIN REMAINING VALUES (MRV)

- Choose variable with minimum number of remaining values in its domain
- Why min rather than max?



MIN REMAINING VALUES (MRV)

- Choose variable with minimum number of remaining values in its domain
- Most constrained variable
- “Fail-fast” ordering



LEAST CONSTRAINING VALUE

- Given choice of variable:
 - Choose least constraining value
 - Aka value that rules out the least values in the remaining variables to be assigned
 - May take some computation to find this
- Why least rather than most?



COST OF BACKTRACKING?

- d values per variable
- n variables
- Possible number of CSP assignments?
- **A) $O(d^n)$**
- B) $O(n^d)$
- C) $O(nd)$



OVERVIEW

- Real world CSPs
- Basic algorithms for solving CSPs
- **Pruning space through propagating information**



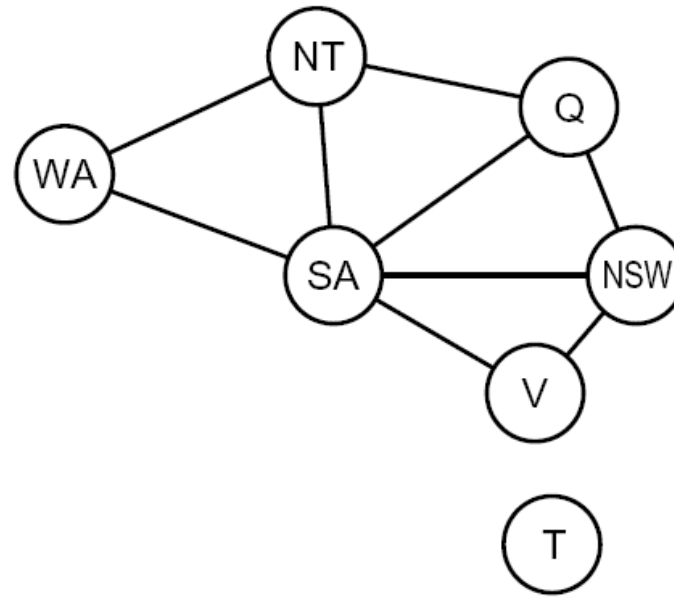
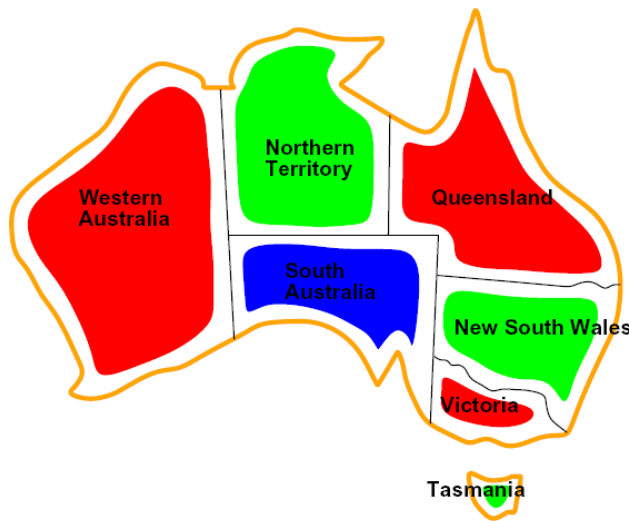
LIMITATIONS OF BACKTRACKING

- Function `Backtracking(csp)` returns soln or fail
 - Return `Backtrack({},csp)`
- Function `Backtrack(assignment,csp)` returns soln or fail
 - If assignment is complete, return assignment
 - $V_i \leftarrow \text{select_unassigned_var}(csp)$
 - For each `val` in `order-domain-values(var,csp,assign)`
 - If value is consistent with assignment
 - Add $[V_i = \text{val}]$ to assignment
 - $\text{Result} \leftarrow \text{Backtrack}(\text{assignment},csp)$
 - If $\text{Result} \neq \text{fail}$, return result
 - Remove $[V_i = \text{val}]$ from assignments
 - Return fail



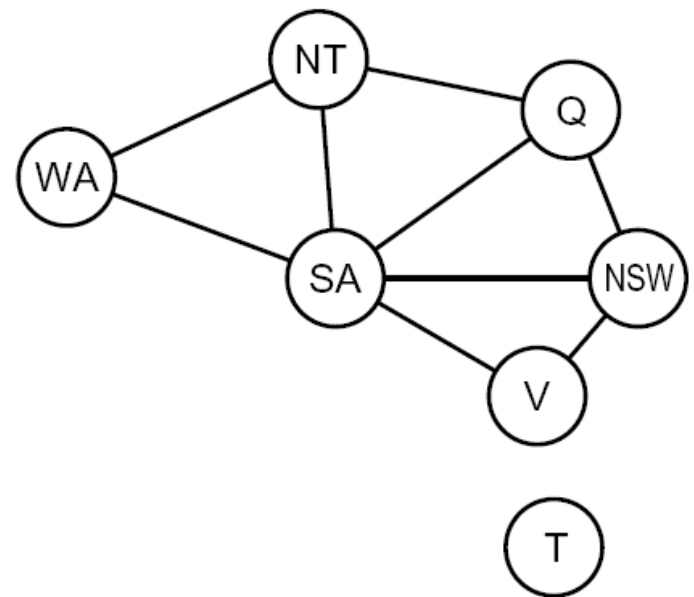
CONSTRAINT GRAPHS

- Nodes are variables
- Arcs show constraints



PROPAGATE INFORMATION

- If we choose a value for one variable, that affects its neighbors
- And then potentially those neighbors...
- Prunes the space of solutions



ARC CONSISTENCY

- Definition:
 - An “arc” (connection between two variables $X \rightarrow Y$ in constraint graph) is consistent if
 - For every value could assign to X
 - There exists some value of Y that could be assigned without violating a constraint



AC-3 (ASSUME BINARY CONSTRAINTS)

- Input: CSP
- Output: CSP, possible with reduced domains for variables, or inconsistent
- Local variables: queue, initially queue of all arcs (binary constraints in csp)
- While queue is not empty
 - $(X_i, X_j) = \text{Remove-First}(\text{queue})$
 - $[\text{domain}X_i, \text{anyChangeToDomain}X_i] = \text{Revise}(\text{csp}, X_i, X_j)$
 - if $\text{anyChangeToDomain}X_i == \text{true}$
 - if $\text{size}(\text{domain}X_i) = 0$, return inconsistent
 - else
 - for each X_k in $\text{Neighbors}(X_i)$ except X_j
 - add (X_k, X_i) to queue
- Return csp

Have to add in
arc for (X_i, X_j) and
 (X_j, X_i)
for i,j constraint



AC-3 (ASSUME BINARY CONSTRAINTS)

- Input: CSP
- Output: CSP, possible with reduced domains for variables, or inconsistent
- Local variables: queue, initially queue of all arcs (binary constraints in csp)
- While queue is not empty
 - $(X_i, X_j) = \text{Remove-First}(\text{queue})$
 - $[\text{domain}X_i, \text{anyChangeToDomain}X_i] = \text{Revise}(\text{csp}, X_i, X_j)$
 - if $\text{anyChangeToDomain}X_i == \text{true}$
 - if $\text{size}(\text{domain}X_i) = 0$, return inconsistent
 - else
 - for each X_k in $\text{Neighbors}(X_i)$ except X_j
 - add (X_k, X_i) to queue
- Return csp

Have to add in
arc for (X_i, X_j) and
 (X_j, X_i)
for i, j constraint

-
- **Function $\text{Revise}(\text{csp}, X_i, X_j)$** returns $\text{Domain}X_i$ and $\text{anyChangeToDomain}X_i$
 - $\text{anyChangeToDomain}X_i = \text{false}$
 - for each x in $\text{Domain}(X_i)$
 - if no value y in $\text{Domain}(X_j)$ allows (x, y) to satisfy constraint between (X_i, X_j)
 - delete x from $\text{Domain}(X_i)$
 - $\text{anyChangeToDomain}X_i = \text{true}$



AC-3 COMPUTATIONAL COMPLEXITY?

- Input: CSP
- Output: CSP, possible with reduced domains for variables, or inconsistent
- Local variables: queue, initially queue of all arcs (binary constraints in csp)
- While queue is not empty
 - $(X_i, X_j) = \text{Remove-First}(\text{queue})$
 - $[\text{domain}X_i, \text{anyChangeToDomain}X_i] = \text{Revise}(\text{csp}, X_i, X_j)$
 - if $\text{anyChangeToDomain}X_i == \text{true}$
 - if $\text{size}(\text{domain}X_i) = 0$, return inconsistent
 - else
 - for each X_k in $\text{Neighbors}(X_i)$ except X_j
 - add (X_k, X_i) to queue
- Return csp

Have to add in arc for (X_i, X_j) and (X_j, X_i) for i,j constraint

D domain values
C binary constraints

Complexity of revise function? D^2

-
- **Function $\text{Revise}(\text{csp}, X_i, X_j)$** returns $\text{Domain}X_i$ and $\text{anyChangeToDomain}X_i$
 - $\text{anyChangeToDomain}X_i = \text{false}$
 - for each x in $\text{Domain}(X_i)$
 - if no value y in $\text{Domain}(X_j)$ allows (x, y) to satisfy constraint between (X_i, X_j)
 - delete x from $\text{Domain}(X_i)$
 - $\text{anyChangeToDomain}X_i = \text{true}$



AC-3 COMPUTATIONAL COMPLEXITY?

- Input: CSP
- Output: CSP, possible with reduced domains for variables, or inconsistent
- Local variables: queue, initially queue of all arcs (binary constraints in csp)
- While queue is not empty
 - $(X_i, X_j) = \text{Remove-First}(\text{queue})$
 - $[\text{domain}X_i, \text{anyChangeToDomain}X_i] = \text{Revise}(\text{csp}, X_i, X_j)$
 - if $\text{anyChangeToDomain}X_i == \text{true}$
 - if $\text{size}(\text{domain}X_i) = 0$, return inconsistent
 - else
 - for each X_k in $\text{Neighbors}(X_i)$ except X_j
 - add (X_k, X_i) to queue
- Return csp

Have to add in arc for (X_i, X_j) and (X_j, X_i) for i,j constraint

D domain values
C binary constraints

Complexity of revise function?
 D^2

-
- **Function $\text{Revise}(\text{csp}, X_i, X_j)$** returns $\text{Domain}X_i$ and $\text{anyChangeToDomain}X_i$
 - $\text{anyChangeToDomain}X_i = \text{false}$
 - for each x in $\text{Domain}(X_i)$
 - if no value y in $\text{Domain}(X_j)$ allows (x, y) to satisfy constraint between (X_i, X_j)
 - delete x from $\text{Domain}(X_i)$
 - $\text{anyChangeToDomain}X_i = \text{true}$

Number of times can put a constraint in queue?



AC-3 COMPUTATIONAL COMPLEXITY?

- Input: CSP
- Output: CSP, possible with reduced domains for variables, or inconsistent
- Local variables: queue, initially queue of all arcs (binary constraints in csp)
- While queue is not empty
 - $(X_i, X_j) = \text{Remove-First}(\text{queue})$
 - $[\text{domain}X_i, \text{anyChangeToDomain}X_i] = \text{Revise}(\text{csp}, X_i, X_j)$
 - if $\text{anyChangeToDomain}X_i == \text{true}$
 - if $\text{size}(\text{domain}X_i) = 0$, return inconsistent
 - else
 - for each X_k in $\text{Neighbors}(X_i)$ except X_j
 - add (X_k, X_i) to queue
- Return csp

Have to add in arc for (X_i, X_j) and (X_j, X_i) for i,j constraint

D domain values
C binary constraints

Complexity of revise function?
 D^2

-
- **Function $\text{Revise}(\text{csp}, X_i, X_j)$** returns $\text{Domain}X_i$ and $\text{anyChangeToDomain}X_i$
 - $\text{anyChangeToDomain}X_i = \text{false}$
 - for each x in $\text{Domain}(X_i)$
 - if no value y in $\text{Domain}(X_j)$ allows (x, y) to satisfy constraint between (X_i, X_j)
 - delete x from $\text{Domain}(X_i)$
 - $\text{anyChangeToDomain}X_i = \text{true}$

Number of times can put a constraint in queue?
D

Total:
 CD^3



SUFFICIENT? CLICK!

- After we run AC-3 have we always found a solution? (aka only 1 value left for each variable)
- A) Yes
- **B) No**



AC-3 EXAMPLE

- Variables: A,B,C,D
- Domain: {1,2,3}
- Constraints: $A \neq B$, $C < B$, $C < D$ (subset of constraints from before)



AC-3 EXAMPLE

- Variables: A,B,C,D
- Domain: {1,2,3}
- Constraints: $A \neq B$, $C < B$, $C < D$ (subset of constraints from before)
- Constraints both ways: $A \neq B$, $B \neq A$, $C < B$, $B > C$, $C < D$, $D > C$



AC-3 EXAMPLE

- Variables: A,B,C,D
- Domain: {1,2,3}
- Constraints: $A \neq B$, $C < B$, $C < D$ (subset of constraints from before)
- Constraints both ways: $A \neq B$, $B \neq A$, $C < B$, $B > C$, $C < D$, $D > C$
- Queue: AB, BA, BC, CB, CD, DC



AC-3 EXAMPLE

- Variables: A,B,C,D
- Domain: {1,2,3}
- Constraints: $A \neq B$, $C < B$, $C < D$ (subset of constraints from before)
- Constraints both ways: $A \neq B$, $B \neq A$, $C < B$, $B > C$, $C < D$, $D > C$
- Queue: AB, BA, BC, CB, CD, DC
- Pop AB:
 - *for each x in $\text{Domain}(A)$*
 - *if no value y in $\text{Domain}(B)$ that allows (x,y) to satisfy constraint between (A,B) , delete x from $\text{Domain}(A)$*
- No change to domain of A



AC-3 EXAMPLE

- Variables: A,B,C,D
- Domain: {1,2,3}
- Constraints: $A \neq B$, $C < B$, $C < D$ (subset of constraints from before)
- Constraints both ways: $A \neq B$, $B \neq A$, $C < B$, $B > C$, $C < D$, $D > C$
- Queue: AB, BA, BC, CB, CD, DC
- Pop AB
- Queue: BA, BC, CB, CD, DC



AC-3 EXAMPLE

- Variables: A,B,C,D
- Domain: {1,2,3}
- Constraints: $A \neq B$, $C < B$, $C < D$ (subset of constraints from before)
- Constraints both ways: $A \neq B$, $B \neq A$, $C < B$, $B > C$, $C < D$, $D > C$
- Queue: AB, BA, BC, CB, CD, DC
- Pop AB
- Queue: BA, BC, CB, CD, DC
- Pop BA
- *for each x in $\text{Domain}(B)$*
 - if no value y in $\text{Domain}(A)$ that allows (x,y) to satisfy constraint between (B,A) , delete x from $\text{Domain}(B)$*
- No change to domain of B



AC-3 EXAMPLE

- Variables: A,B,C,D
- Domain: {1,2,3}
- Constraints: $A \neq B$, $C < B$, $C < D$ (subset of constraints from before)
- Constraints both ways: $A \neq B$, $B \neq A$, $C < B$, $B > C$, $C < D$, $D > C$
- Queue: AB, BA, BC, CB, CD, DC
- Queue: BA, BC, CB, CD, DC
- Queue: BC, CB, CD, DC
- Pop BC
- *for each x in Domain(B)*
 - if no value y in Domain(C) that allows (x,y) to satisfy constraint between (B,C), delete x from Domain(B)*
- If B is 1, constraint $B > C$ cannot be satisfied. So delete 1 from B's domain, $B=\{2,3\}$
- **Also have to add neighbors of B (except C) back to queue: AB**
- Queue: AB, CB, CD, DC



AC-3 EXAMPLE

Variables: A,B,C,D

Domain: {1,2,3}

Constraints: $A \neq B$, $C < B$, $C < D$

- Queue: AB, BA, BC, CB, CD, DC $A-D = \{1,2,3\}$
- Queue: BA, BC, CB, CD, DC $A-D = \{1,2,3\}$
- Queue: BC, CB, CD, DC $A-D = \{1,2,3\}$
- Queue: AB, CB, CD, DC $B=\{2,3\}$, $A/C/D = \{1,2,3\}$
- Pop AB
 - For every value of A is there a value of B such that $A \neq B$?
 - Yes, so no change



AC-3 EXAMPLE

Variables: A,B,C,D

Domain: {1,2,3}

Constraints: $A \neq B$, $C < B$, $C < D$

- Queue: AB, BA, BC, CB, CD, DC, $A-D = \{1,2,3\}$
- Queue: BA, BC, CB, CD, DC $A-D = \{1,2,3\}$
- Queue: BC, CB, CD, DC $A-D = \{1,2,3\}$
- Queue: AB, CB, CD, DC $B=\{2,3\}$, $A/C/D = \{1,2,3\}$
- Queue: CB, CD, DC $B=\{2,3\}$, $A/C/D = \{1,2,3\}$
- Pop CB
 - For every value of C is there a value of B such that $C < B$
 - If $C = 3$, no value of B that fits
 - So delete 3 from C's domain, $C = \{1,2\}$
 - **Also have to add neighbors of C (except B) back to queue: no change because already in**



AC-3 EXAMPLE

Variables: A,B,C,D

Domain: {1,2,3}

Constraints: $A \neq B$, $C < B$, $C < D$

- Queue: AB, BA, BC, CB, CD, DC, $A-D = \{1,2,3\}$
- Queue: BA, BC, CB, CD, DC $A-D = \{1,2,3\}$
- Queue: BC, CB, CD, DC $A-D = \{1,2,3\}$
- Queue: AB, CB, CD, DC $B=\{2,3\}$, $A/C/D = \{1,2,3\}$
- Queue: CB, CD, DC $B=\{2,3\}$, $A/C/D = \{1,2,3\}$
- Queue: CD, DC $B=\{2,3\}$, $C = \{1,2\}$, $A,D = \{1,2,3\}$
- Pop CD
 - For every value of C, is there a value of D such that $C < D$?
 - Yes, so no change



AC-3 EXAMPLE

Variables: A,B,C,D

Domain: {1,2,3}

Constraints: $A \neq B$, $C < B$, $C < D$

- Queue: AB, BA, BC, CB, CD, DC $A-D = \{1,2,3\}$
- Queue: BA, BC, CB, CD, DC $A-D = \{1,2,3\}$
- Queue: BC, CB, CD, DC $A-D = \{1,2,3\}$
- Queue: AB, CB, CD, DC $B=\{2,3\}$, $A/C/D = \{1,2,3\}$
- Queue: CB, CD, DC $B=\{2,3\}$, $A/C/D = \{1,2,3\}$
- Queue: CD, DC $B=\{2,3\}$, $C = \{1,2\}$ $A,D = \{1,2,3\}$
- Queue: DC $B=\{2,3\}$, $C = \{1,2\}$ $A,D = \{1,2,3\}$
- For every value of D is there a value of C such that $D > C$?
 - Not if $D = 1$
 - So $D = \{2,3\}$



AC-3 EXAMPLE

Variables: A,B,C,D

Domain: {1,2,3}

Constraints: $A \neq B$, $C < B$, $C < D$

- Queue: AB, BA, BC, CB, CD, DC $A-D = \{1,2,3\}$
- Queue: BA, BC, CB, CD, DC $A-D = \{1,2,3\}$
- Queue: BC, CB, CD, DC $A-D = \{1,2,3\}$
- Queue: AB, CB, CD, DC $B=\{2,3\}$, $A/C/D = \{1,2,3\}$
- Queue: CB, CD, DC $B=\{2,3\}$, $A/C/D = \{1,2,3\}$
- Queue: CD, DC $B=\{2,3\}$, $C = \{1,2\}$ $A,D = \{1,2,3\}$
- Queue: DC $B=\{2,3\}$, $C = \{1,2\}$ $A,D = \{1,2,3\}$
- $A = \{1,2,3\}$ $B=\{2,3\}$, $C = \{1,2\}$ $D = \{2,3\}$



FORWARD CHECKING

- When assign a variable, make all of its neighbors arc-consistent



BACKTRACKING + FORWARD CHECKING

- Function *Backtrack(assignment, csp)* returns soln or fail
 - If assignment is complete, return assignment
 - $V_i \leftarrow \text{select_unassigned_var}(csp)$
 - For each val in *order-domain-values(var, csp, assign)*
 - If value is consistent with assignment
 - Add $[V_i = \text{val}]$ to assignment
 - Make domains of all neighbors of V_i arc-consistent with $[V_i = \text{val}]$**
 - Result $\leftarrow \text{Backtrack}(\text{assignment}, csp)$
 - If Result \neq fail, return result
 - Remove $[V_i = \text{val}]$ from assignments
 - Return fail



MAINTAINING ARC CONSISTENCY

- Forward checking doesn't ensure all arcs are consistent
- AC-3 detects failure faster than forward checking
- What's the downside? **Computation**



MAINTAINING ARC CONSISTENCY (MAC)

- Function *Backtrack(assignment, csp)* returns soln or fail
 - If assignment is complete, return assignment
 - $V_i \leftarrow \text{select_unassigned_var}(csp)$
 - For each val in *order-domain-values(var, csp, assign)*
 - If value is consistent with assignment
 - Add $[V_i = \text{val}]$ to assignment
 - Run AC-3 to make all variables arc-consistent with $[V_i = \text{val}]$.**
 - Initial queue is arcs (X_j, V_i) of neighbors of V_i that are unassigned, but add other arcs if these vars change domains.**
 - Result $\leftarrow \text{Backtrack}(assignment, csp)$**
 - If Result \neq fail, return result
 - Remove $[V_i = \text{val}]$ from assignments
 - Return fail



SUFFICIENT TO AVOID BACKTRACKING?

- If we maintain arc consistency, we will never have to backtrack while solving a CSP
- A) True
- **B) False**



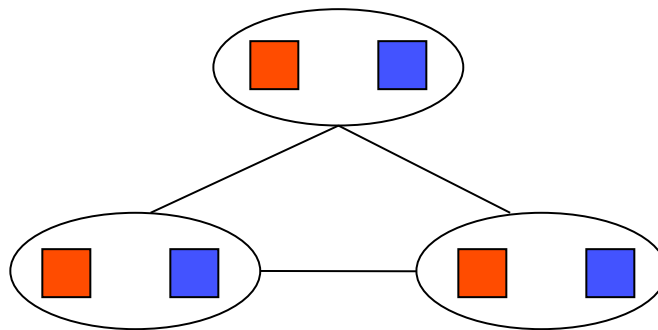
AC LIMITATIONS

- After running AC-3
 - Can have one solution left
 - Can have multiple solutions left
 - Can have no solutions left (and not know it)



AC LIMITATIONS

- After running AC-3
 - Can have one solution left
 - Can have multiple solutions left
 - Can have no solutions left (and not know it)



*What went
wrong here?*



SUMMARY

- Be able to define real world CSPs
- Understand basic algorithm (backtracking)
 - Complexity relative to basic search algorithms
 - Doesn't require problem specific heuristics
 - Ideas shaping search (LCV, etc)
- Pruning space through propagating information
 - Arc consistency
 - Tradeoffs: + reduces search space,
- costs computation
- Relevant reading: Chapter 6

