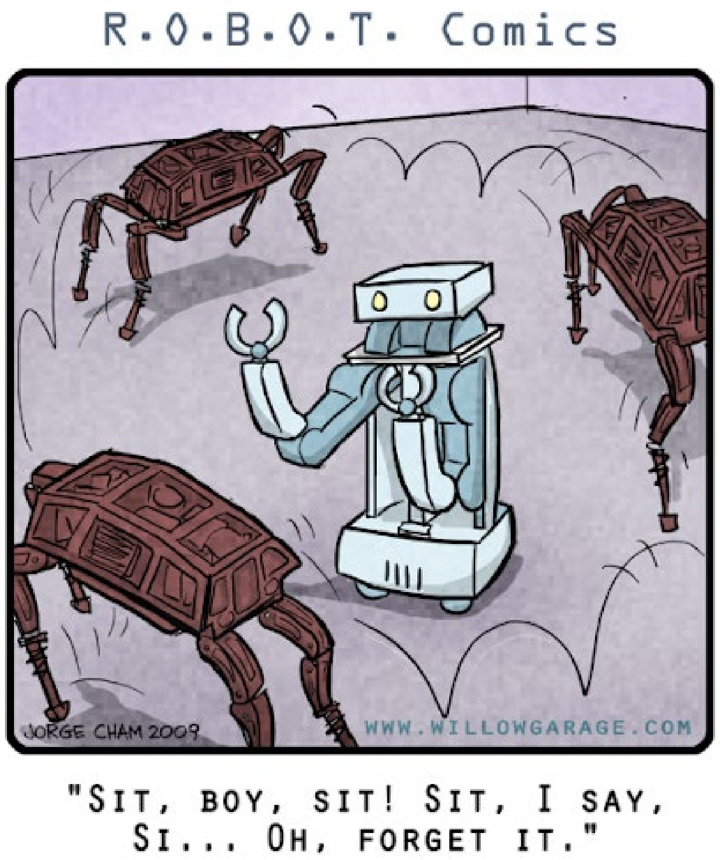


CS 4649/7649 Robot Intelligence: Planning

Constraints II: CSP Methods & Complexity

CS 4649/7649 – Asst. Prof. Matthew Gombolay

Slides adapted from:
16.410 Brian Williams
Dechter 1991
Russell and Norvig AIMA

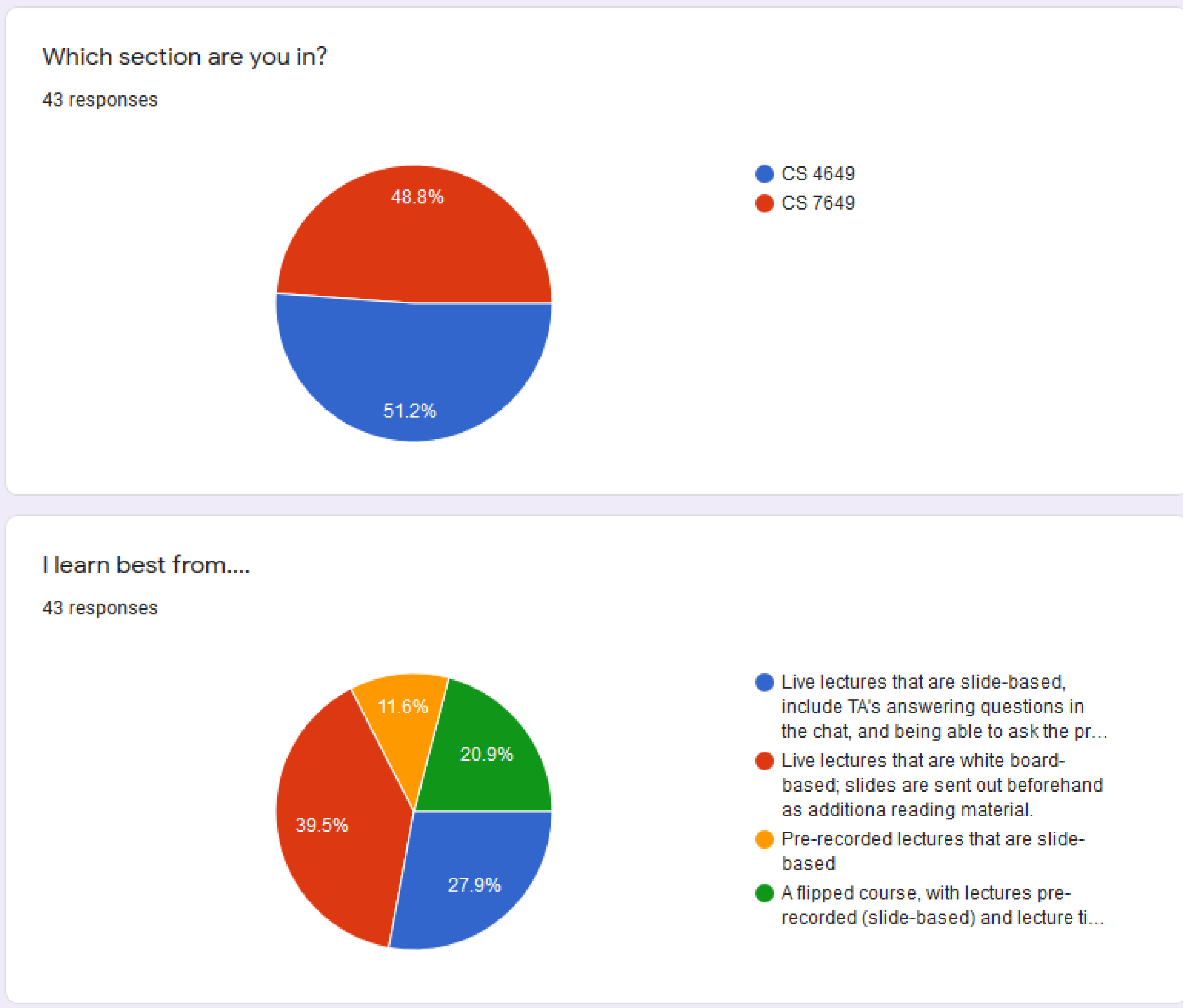
Assignments

- Due Wednesday, 9/02
  - Pset 2 due at 11:59 PM Eastern
- Due Monday, 9/07
  - **Labor Day Holiday**
- Due Wednesday, 9/9
  - Pset 3 due at 11:59 PM Eastern

Course Format

In CS 4649/7649, you've now had a chance to experience live and pre-recorded lectures that are slide-based, live lectures that are white board-based, and live (unrecorded) office hours, and interactions on Piazza. We want to take stock of which learning modes are most effective for this course during the COVID-19 pandemic. Thank you for your responses!

https://forms.gle/87PS6fPGC6EXkuN67



(Recap) Constraint Satisfaction Problems (CSP)

Input: A CSP is a 3-tuple (i.e., triple) $\langle V, D, C\rangle$ where:

- V is a set of variables V_i
- D is a set of variable domains,
  - The domain of variable V_i is denoted D_i
- C is the set of constraints on assignments to V
  - Each constraint $C_j = \langle S_j, R_j \rangle$ specifies allowed variable assignments
  - S_j, the constraint's scope, is a subset of variables V
  - R_j, the constraint's relation, is a set of assignments to S_j

Output: A full assignment to V from elements of D such that all constraints C are satisfied.

Constraint Modeling (Programming) Languages

Features: Declarative specification of the problem that separates the formulation and the search strategy.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

A sudoku puzzle...

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

...and its solution numbers marked in red

Sudoko puzzle (left) and solutions (right)

Source: http://www.comp.nus.edu.sg/cs1101x/3.ca/labs/07s1/lab7/img/

Outline

- Analysis of constraint propagation
- Solving CSPs using Search

What is the Complexity of AC-1

AC-1 (CSP)

Input: CSP = $\langle X, D, C \rangle$

Output: CSP’, the largest arc-consistent subset of CSP

1. WHILE (domains are being changed)
2. FOR every $C_{ij} \in C$
3. Revise(x_i, x_j)
4. Revise(x_j, x_i)
5. ENDFOR
6. ENDWHILE

Assume:

- There are n variables
- Domains are of size at most k
- There are e binary constraints

What is the Complexity of AC-1?

Assume:

- There are n variables
- Domains are of size at most $k = \max_i |D_i|$
- There are e binary constraints

Which is the correct complexity?

1. $O(k^2)$
2. $O(enk^2)$
3. $O(enk^3)$
4. $O(nek)$

What is the Complexity of AC-1?

AC-1 (CSP)

Input: CSP = $\langle X, D, C \rangle$

Output: CSP', the largest arc-consistent subset of CSP

WHILE (domains are being changed)
    FOR every $C_{ij} \in C$
        Revise(x_i, x_j)
        Revise(x_j, x_i)
    ENDFOR
ENDWHILE

Assume:

- There are n variables
- Domains are of size at most k
- There are e binary constraints

What is the Complexity of AC-1?

AC-1 (CSP)
Input: CSP = $\langle X, D, C \rangle$
Output: CSP', the largest arc-consistent subset of CSP

1. WHILE (domains are being changed)
2. FOR every $C_{ij} \in C$
3. Revise(x_i, x_j)
4. Revise(x_j, x_i)
5. ENDFOR
6. ENDWHILE

Why?

$O(nk) * (\ldots$

$\ldots)$

Assume:
- There are n variables
- Domains are of size at most k
- There are e binary constraints

Proof Sketch [By Deduction]:
1. Line 1 only iterates if we deleted something from a domain
2. The number of possible domain's we could modify is n
3. The number of possible domain changes we could make to each domain is less than or equal to k
4. Therefore, we iterate at most nk times

What is the Complexity of AC-1?

AC-1 (CSP)

Input: CSP = $\langle X, D, C \rangle$

Output: CSP', the largest arc-consistent subset of CSP

1. WHILE (domains are being changed) $O(nk) * (...$
2. FOR every $C_{ij} \in C$ $O(e) * (...$
3. Revise(x_i, x_j)
4. Revise(x_j, x_i)
5. ENDFOR $...)$
6. ENDWHILE $...)$

Assume:

- There are n variables
- Domains are of size at most k
- There are e binary constraints

What is the Complexity of AC-1?

AC-1 (CSP)

Input: CSP = $\langle X, D, C\rangle$

Output: CSP', the largest arc-consistent subset of CSP

1. WHILE (domains are being changed) $O(nk) * (\ldots$
2. FOR every $C_{ij} \in C$ $O(e) * (\ldots$
3. Revise(x_i, x_j)
4. Revise(x_j, x_i)
5. ENDFOR $\ldots)$
6. ENDWHILE $\ldots)$

What is the complexity of REVISE(,)?

Assume:

- There are n variables
- Domains are of size at most k
- There are e binary constraints

Revise: A directed arc consistency procedure

Revise(x_i, x_j)

Input: Variables x_i and x_j with domains D_i and D_j and constraint relation R_{ij}

Output: Pruned D_i such that x_i is directed arc-consistent relative to x_j

1. FOR each $a_i \in D_i$
2.   IF there is no $a_j \in D_j$ such that $\langle a_i, a_j \rangle \in R_{ij}$ THEN
3.   Delete a_i from D_i
4.   ENDIF
5. ENDFOR

Revise: A directed arc consistency procedure

Revise(x_i, x_j)

Input: Variables x_i and x_j with domains D_i and D_j and constraint relation R_{ij}

Output: Pruned D_i such that x_i is directed arc-consistent relative to x_j

1. FOR each $a_i \in D_i$ $O(k) * (\ldots$
2. IF there is no $a_j \in D_j$ such that $\langle a_i, a_j \rangle \in R_{ij}$ THEN
3. Delete a_i from D_i
4. ENDIF
5. ENDFOR $\ldots)$

Revise: A directed arc consistency procedure

Revise(x_i, x_j)

Input: Variables x_i and x_j with domains D_i and D_j and constraint relation R_{ij}

Output: Pruned D_i such that x_i is directed arc-consistent relative to x_j

1. FOR each $a_i \in D_i$ $O(k) * (\ldots$
2. IF there is no $a_j \in D_j$ such that $\langle a_i, a_j \rangle \in R_{ij}$ THEN $O(k) * (\ldots$
3. Delete a_i from D_i
4. ENDIF $\ldots)$
5. ENDFOR $\ldots)$

Complexity of Revise()?

$= O(k^2)$

What is the Complexity of AC-1?

AC-1 (CSP)
Input: CSP = $\langle X, D, C \rangle$
Output: CSP', the largest arc-consistent subset of CSP

1. WHILE (domains are being changed) $O(nk) * (\ldots$
2. FOR every $C_{ij} \in C$ $O(e)$
3. Revise(x_i, x_j)
4. Revise(x_j, x_i)
5. ENDFOR $\ldots)$
6. ENDWHILE $\ldots)$

What is the complexity of REVISE(,)?

Assume:

- There are n variables
- Domains are of size at most k
- There are e binary constraints

What is the Complexity of AC-1

AC-1 (CSP)

Input: CSP = $\langle X, D, C \rangle$

Output: CSP', the largest arc-consistent subset of CSP

1. WHILE (domains are being changed) $O(nk) * (\ldots$
2. FOR every $C_{ij} \in C$ $O(e) * (\ldots$
3. Revise(x_i, x_j) $(O(k^2)$
4. Revise(x_j, x_i) $+O(k^2))$
5. ENDFOR $\ldots)$
6. ENDWHILE $\ldots)$

Complexity of AC-1?

$$= O(nk * e * k^2)$$
$$= (enk^3)$$

What is the Complexity of AC-1

Assume:

- There are n variables
- Domains are of size at most k
- There are e binary constraints

Which is the correct complexity?

1. $O(k^2)$
2. $O(enk^2)$
3. $\boldsymbol{O(enk^3)}$
4. $O(nek)$

Full Arc-Consistency via AC-3 (Waltz CP)

AC-3 (CSP)

Input: CSP = $\langle X, D, C \rangle$

Output: CSP', the largest arc-consistent subset of CSP

1. FOR every $C_{ij} \in C$
2. $\quad Q \leftarrow Q \cup \{\langle x_i, x_j \rangle, \langle x_j, x_i \rangle\}$
3. ENDFOR
4. While $Q \neq \emptyset$
5. $\quad$ Select and delete arc $\langle x_i, x_j \rangle$ from Q
6. $\quad$ Revise(x_i, x_j)
7. $\quad$ IF Revise(x_i, x_j) caused a change to D_i
8. $\quad\quad Q \leftarrow Q \cup \{\langle x_k, x_i \rangle | k \neq i, k \neq j\}$
9. $\quad$ ENDIF
10. ENDWHILE

Full Arc-Consistency via AC-3 (Waltz CP)

AC-3 (CSP)

Input: CSP = $\langle X, D, C \rangle$

Output: CSP’, the largest arc-consistent subset of CSP

FOR every $C_{ij} \in C$

    $Q \leftarrow Q \cup \{\langle x_i, x_j \rangle, \langle x_j, x_i \rangle\}$

ENDFOR

While $Q \neq \emptyset$

    Select and delete arc $\langle x_i, x_j \rangle$ from Q

    Revise(x_i, x_j)

    IF Revise(x_i, x_j) caused a change to D_i

        $Q \leftarrow Q \cup \{\langle x_k, x_i \rangle | k \neq i, k \neq j\}$

    ENDIF

ENDWHILE

Full Arc-Consistency via AC-3 (Waltz CP)

AC-3 (CSP)

Input: CSP = $\langle X, D, C \rangle$

Output: CSP’, the largest arc-consistent subset of CSP

1. FOR every $C_{ij} \in C$ $\qquad O(e)$ +
2. $\quad Q \leftarrow Q \cup \{\langle x_i, x_j \rangle, \langle x_j, x_i \rangle\}$
3. ENDFOR
4. While $Q \neq \emptyset$
5. $\quad$ Select and delete arc $\langle x_i, x_j \rangle$ from Q
6. $\quad$ Revise(x_i, x_j)
7. $\quad$ IF Revise(x_i, x_j) caused a change to D_i
8. $\qquad Q \leftarrow Q \cup \{\langle x_k, x_i \rangle | k \neq i, k \neq j\}$
9. $\quad$ ENDIF
10. ENDWHILE

 # Iterations of while loop determined by # of times line 7 is TRUE (as well as e, k, and n).

Full Arc-Consistency via AC-3 (Waltz CP)

AC-3 (CSP)

Input: CSP = $\langle X, D, C \rangle$

Output: CSP’, the largest arc-consistent subset of CSP

1. FOR every $C_{ij} \in C$ $\quad O(e) + \cdots$
2. $\quad Q \leftarrow Q \cup \{\langle x_i, x_j \rangle, \langle x_j, x_i \rangle\}$
3. ENDFOR
4. While $Q \neq \emptyset$
5. $\quad$ Select and delete arc $\langle x_i, x_j \rangle$ from Q
6. $\quad$ Revise(x_i, x_j) $\quad O(k^2)$
7. $\quad$ IF Revise(x_i, x_j) caused a change to D_i
8. $\quad\quad Q \leftarrow Q \cup \{\langle x_k, x_i \rangle | k \neq i, k \neq j\}$
9. $\quad$ ENDIF
10. ENDWHILE

 # Iterations of while loop determined by # of times line 7 is TRUE (as well as e, k, and n).

Full Arc-Consistency via AC-3 (Waltz CP)

AC-3 (CSP)

Input: CSP = $\langle X, D, C \rangle$

Output: CSP', the largest arc-consistent subset of CSP

1. FOR every $C_{ij} \in C$ $\quad O(e) + \cdots$
2. $\quad Q \leftarrow Q \cup \{\langle x_i, x_j \rangle, \langle x_j, x_i \rangle\}$
3. ENDFOR
4. While $Q \neq \emptyset$
5. $\quad$ Select and delete arc $\langle x_i, x_j \rangle$ from Q
6. $\quad$ Revise(x_i, x_j) $\quad O(k^2)$
7. $\quad$ IF Revise(x_i, x_j) caused a change to D_i $\quad * O(ek)$
8. $\quad\quad Q \leftarrow Q \cup \{\langle x_k, x_i \rangle | k \neq i, k \neq j\}$
9. $\quad$ ENDIF
10. ENDWHILE

 # Iterations of while loop determined by # of times line 7 is TRUE (as well as e, k, and n).

Why?

PSet 3

Complexity of AC-3?

$$= O(e + ek * k^2) = O(ek^3)$$

Is arc consistency sound and complete?

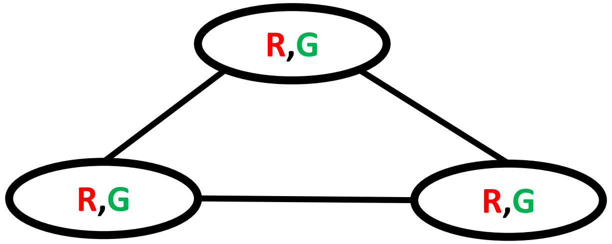
An arc consistent solution selects a value for every variable from its arc consistent domain.

Soundness: All solutions to the CSP are arc consistent solutions?

- Yes
- No

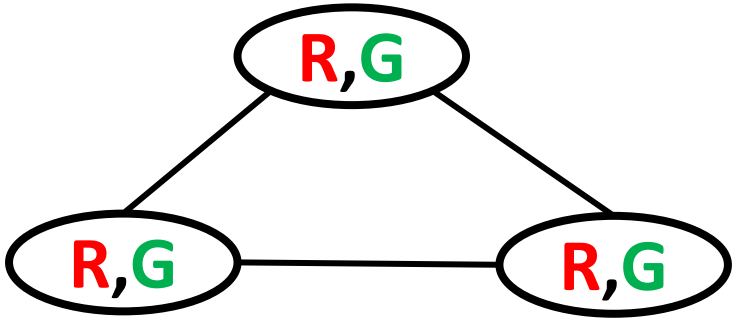
Completeness: All arc-consistent solutions are solutions to the CSP?

- Yes
- No

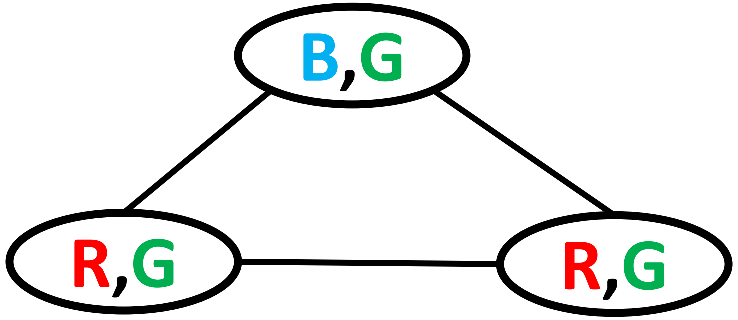


Incomplete: Arc consistency doesn't rule out all infeasible solutions

Graph Coloring Problem



Arc consistent, but no solutions



Arc consistent, but 2 solutions, not 8.

B, R, G
B, G, R

To solve CSPs, we combine

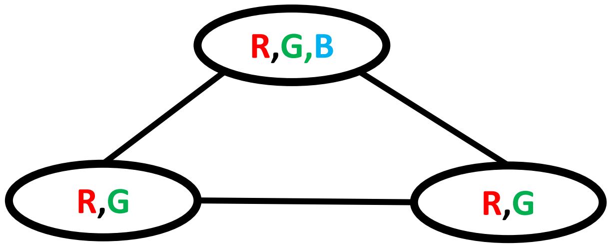
1. Arc consistency (constraint propagation),
    - Eliminates values that are shown locally to not be a part of any solution
2. Search
    - Explores consequences of committing to particular assignments
    - Methods incorporating search:
        - Standard Search
        - Backtrack Search (BT)
        - BT with Forward Checking (FC)
        - Dynamic Variable Ordering (DVO)
        - Iterative Repair
        - Back jumping (BJ)

To solve CSPs, we combine

1. Arc consistency (constraint propagation),
    - Eliminates values that are shown locally to not be a part of any solution
2. Search
    - Explores consequences of committing to particular assignments
    - Methods incorporating search:
        - Standard Search
        - Backtrack Search (BT)
        - BT with Forward Checking (FC)
        - Dynamic Variable Ordering (DVO)
        - Iterative Repair
        - Back jumping (BJ)

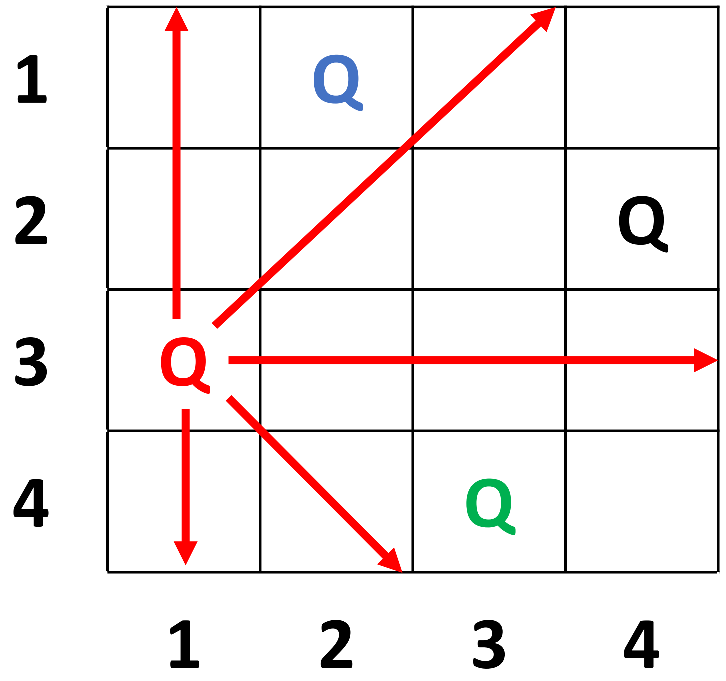
Solving CSPs using Generic Search

- State
  - Partial assignment to variables, made thus far.
- Initial State
  - No assignment.
- Operator
  - Creates new assignment ($X_i = v_{ij}$)
    - Select any unassigned variable X_i
    - Select any one of its domain values v_{ij}
  - Child extends parent assignments with new.
- Goal Test
  - All variables are assigned.
  - All constraints are satisfied.



- Branching factor?
  Sum of domain size of all variables $O(|V||D|)$
- Performance?
  Exponential in the branching factor $O\left((|V||D|)^{(|V||D|)}\right)$

Search Performance on N Queens



- **Standard Search** // A handful of queens
- **Backtracking** // About 15 queens

Solving CSPs with Standard Search

- Standard Search:
  - Children select any value for any variable [O(|v|*|d|)].
  - Test complete assignments for consistency against CSP.

- Observations:

1. The order in which variables are assigned does not change the solution.
   - Many paths denote the same solution, (|v|!),

   → Expand only one path (i.e., use one variable ordering).

2. We can identify a dead end before we assign all variables.
   - Extensions to inconsistent partial assignments are always inconsistent

   → Check consistency after each assignment

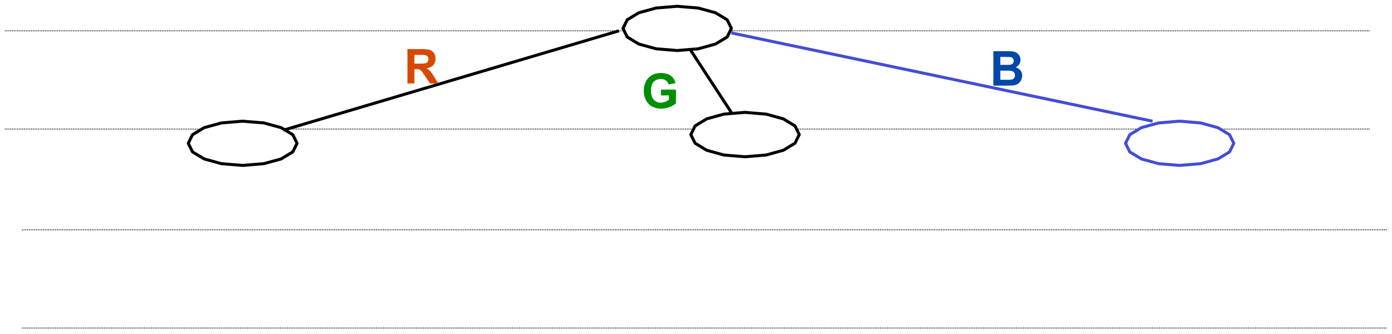
Backtrack Search (BT)

1. Expand assignments of one variable at each step.
2. Pursue depth first.
3. Check consistency after each expansion, and backup.

V_1 assignments

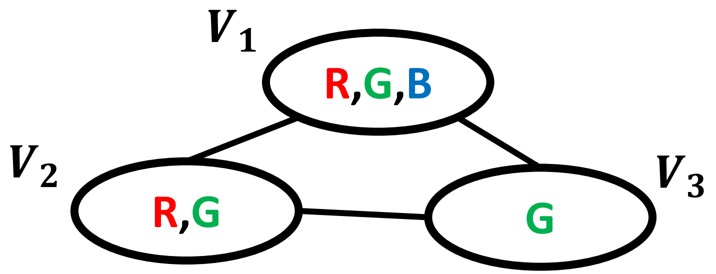
V_2 assignments

V_3 assignments



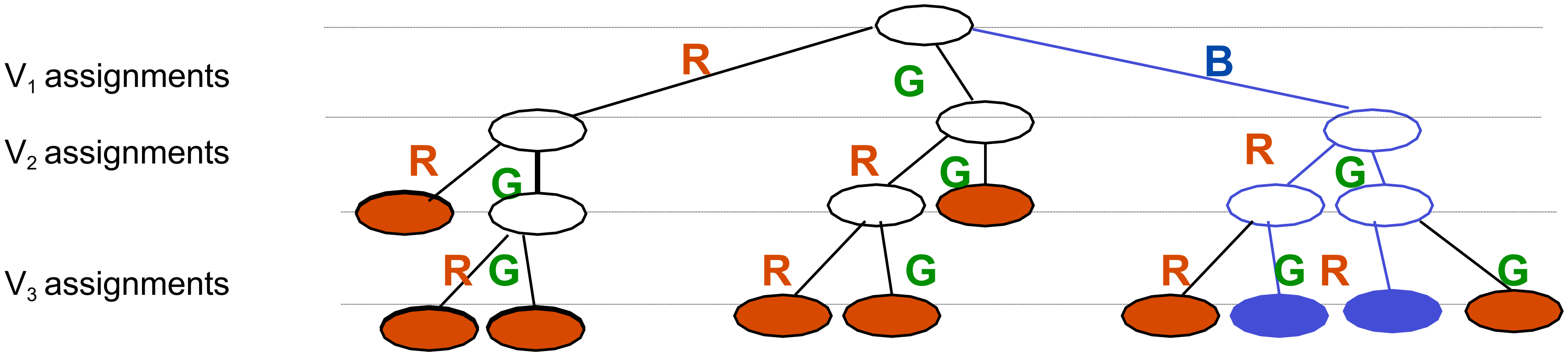
Preselect order of variables to assign

Assign designated variable



Backtrack Search (BT)

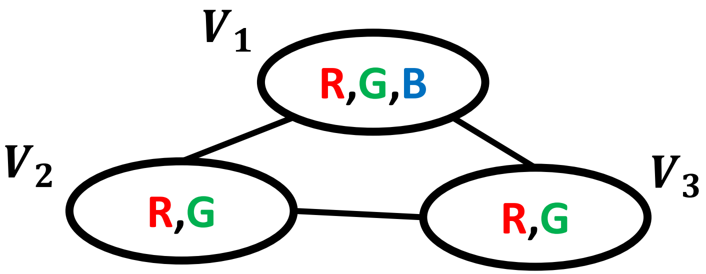
1. Expand assignments of one variable at each step.
2. Pursue depth first.
3. Check consistency after each expansion, and backup.



Preselect order of variables to assign

Assign designated variable

Backup at inconsistent assignment



Procedure Backtracking

Input: A constraint network R = <X, D, C>

Output: A solution, or notification that the network is inconsistent.

$i \leftarrow 1; a_i = \{\}$	Initialize variable counter, assignments
$D'_i \leftarrow D_i$;	Copy domain of first variable.
while $1 \le i \le n$	
instantiate $x_i \leftarrow$ Select-Value();	Add to assignments a_i
if x_i is null	No value was returned,
$i \leftarrow i - 1$;	then backtrack
else	
$i \leftarrow i + 1$;	Else step forward and
$D'_i \leftarrow D_i$;	Copy domain of next variable
end while	
if $i = 0$	
return "inconsistent"	
else	
return $\vec{a_i}$, the instantiated values of $\{x_i, \ldots, x_n\}$	
end procedure	

Procedure Select-Value()

Output: A value in D'_i consistent with a_{i-1}, or null, if none.

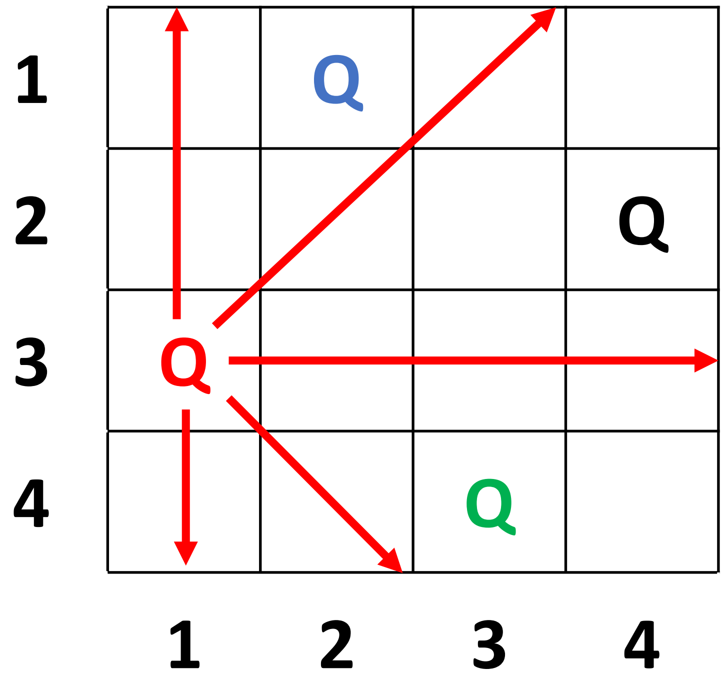
while D'_i is not empty
    select an arbitrary element $a \in D'_i$ and remove a from D'_i
    if consistent(a_{i-1}, $x_i = a$)
        return *a;*
end while
return null   //no consistent value
end procedure

Constraint Processing,

by R. Dechter

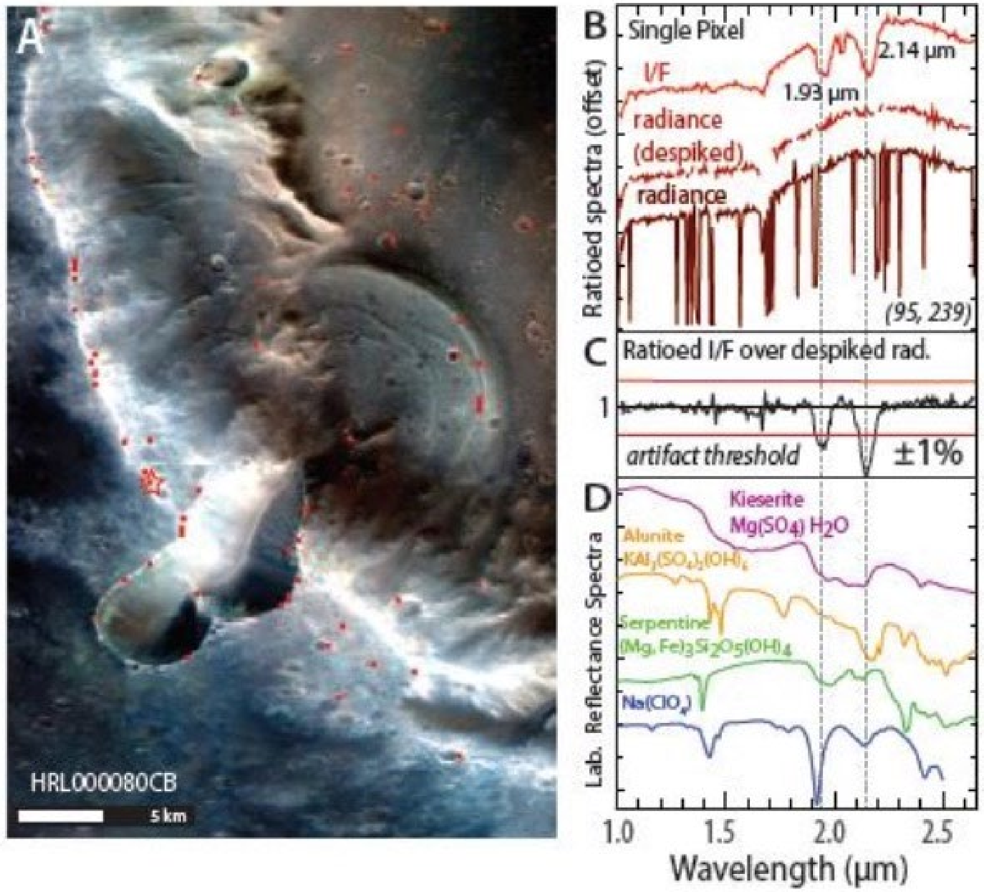
pgs 123-127

Search Performance on N Queens

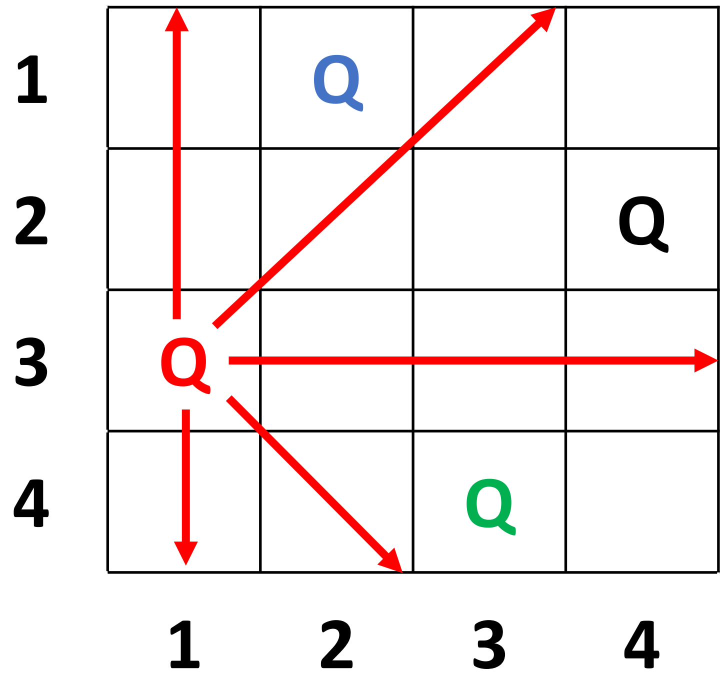


- **Standard Search** // A handful of queens
- **Backtracking** // About 15 queens

Mid-lecture break



Search Performance on N Queens



- **Standard Search** // A handful of queens
- **Backtracking** // About 15 queens
- **BT with Forward Checking** // About 30 queens

Combining Backtracking and Limited Constraint Propagation

Initially: Prune domains using constraint propagation (optional) Loop:

- If complete consistent assignment, then return it, Else…
- Choose unassigned variable.
- Choose assignment from variable's pruned domain.
- Prune (some) domains using Revise (i.e., arc-consistency).
- If a domain has no remaining elements, then backtrack.

Question: Full propagation is $O(ek^3)$, how much propagation should we do?

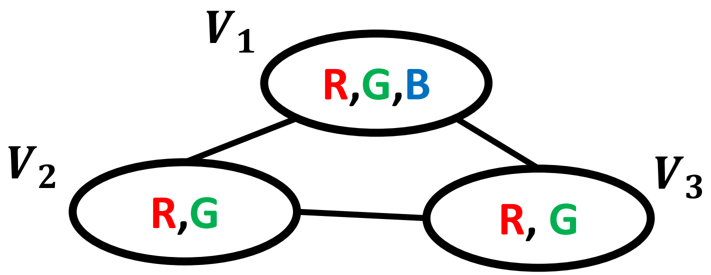
Very little (except for big problems)

Forward Checking (FC)

- Check arc consistency ONLY for arcs that terminate on the new assignment [$O(e\ k)$ total].

Backtracking with Forward Checking

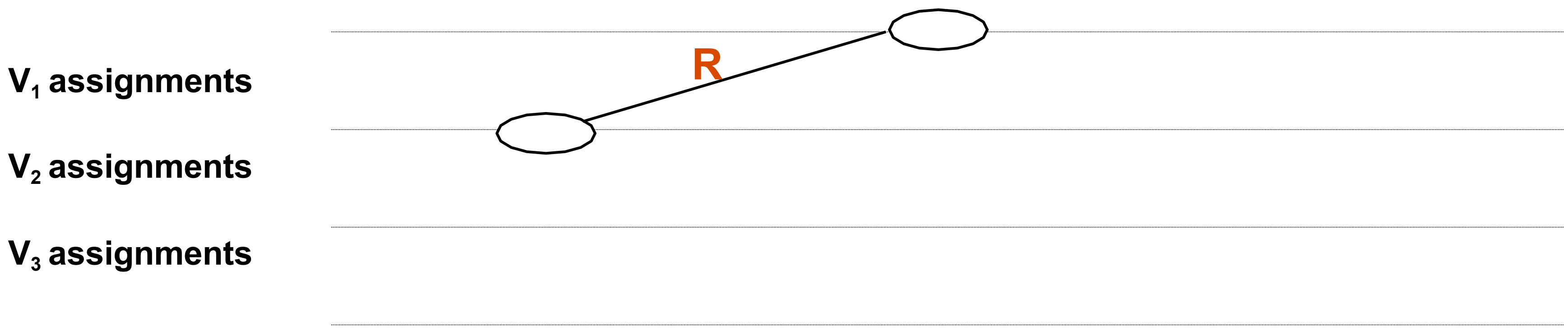
2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.

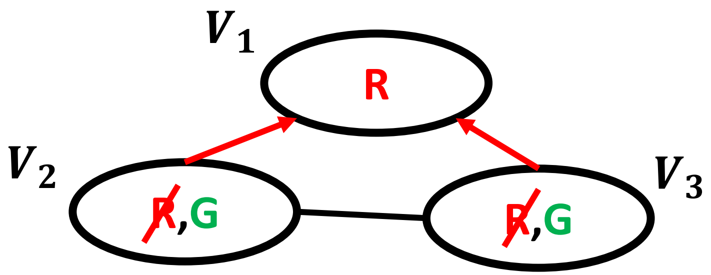


1. Perform initial pruning.

Backtracking with Forward Checking

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.

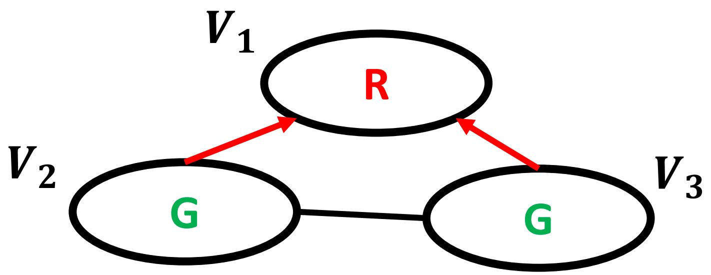




1. Perform initial pruning.

Backtracking with Forward Checking

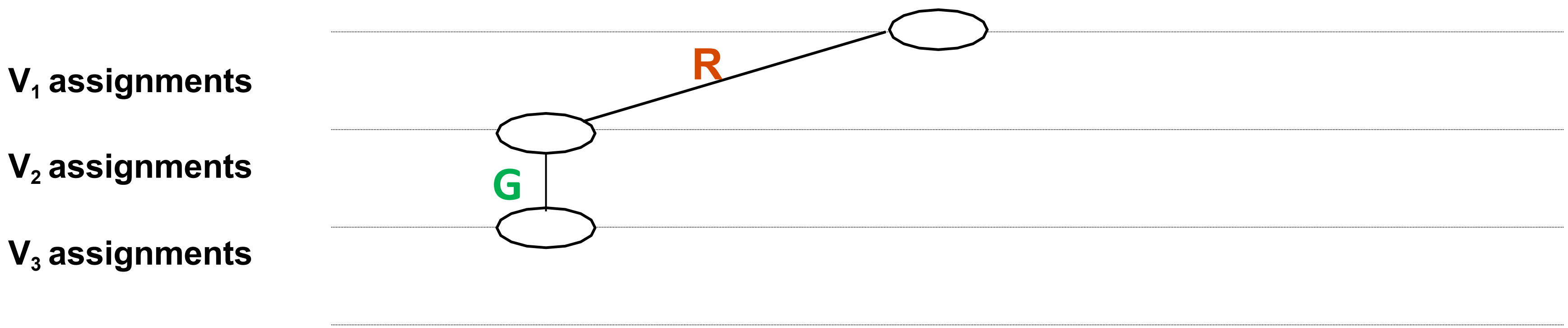
2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.

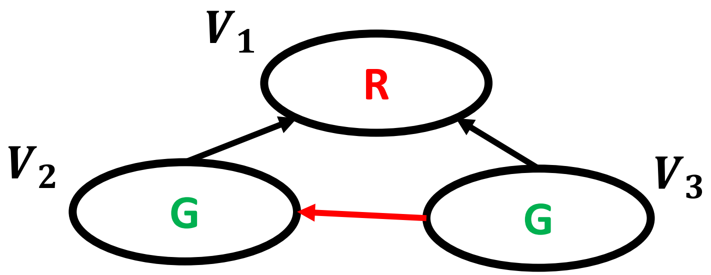


1. Perform initial pruning.

Backtracking with Forward Checking

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.



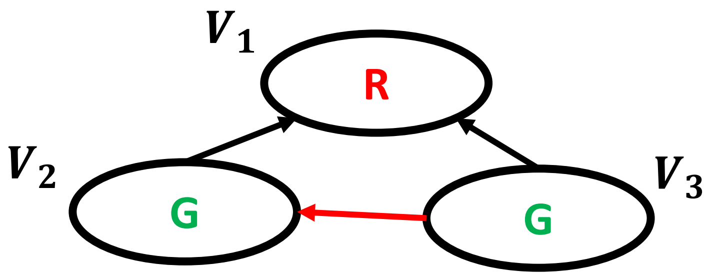


Note: No need to check new assignment against previous assignments

1. Perform initial pruning.

Backtracking with Forward Checking

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.

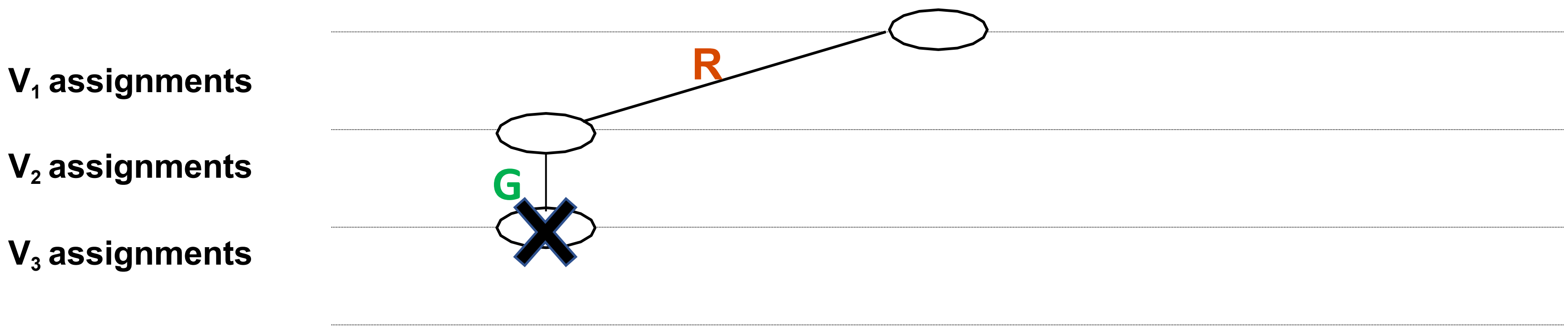


Note: No need to check new assignment against previous assignments

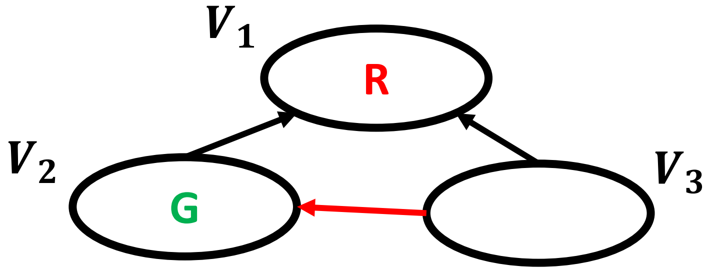
1. Perform initial pruning.

Backtracking with Forward Checking

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.



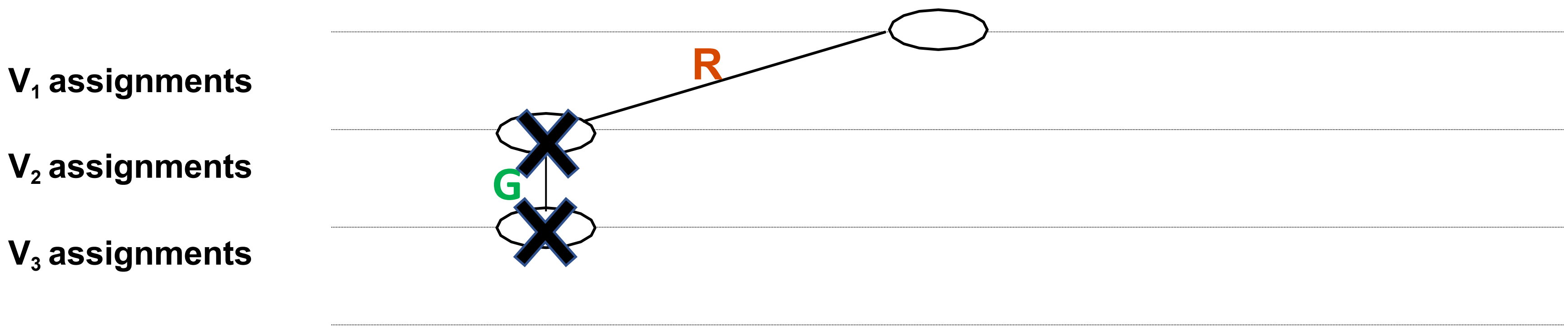
3. We have a conflict whenever a domain becomes empty.
→ Backtrack



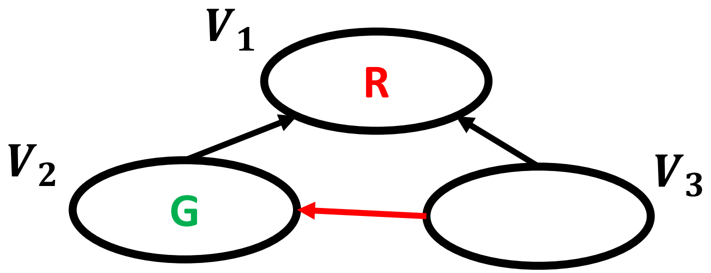
1. Perform initial pruning.

Backtracking with Forward Checking

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.



3. We have a conflict whenever a domain becomes empty.
→ Backtrack



1. Perform initial pruning.

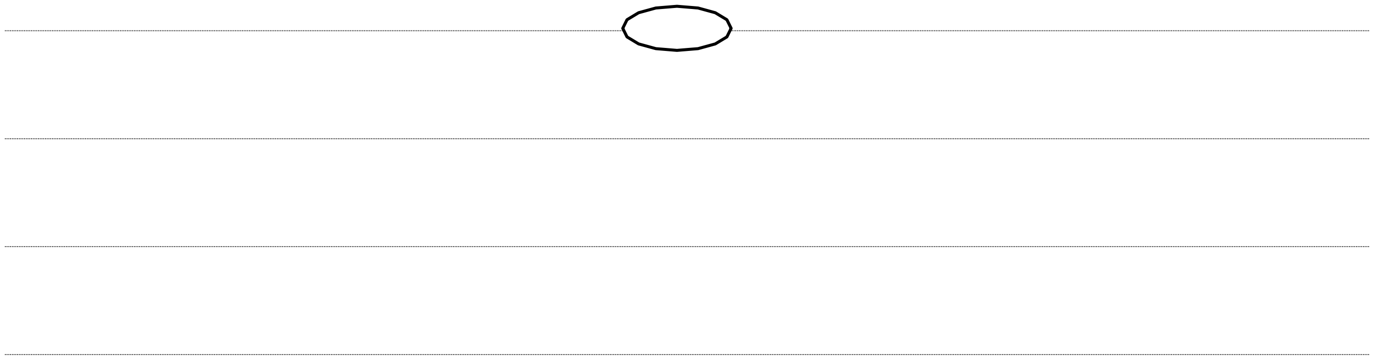
Backtracking with Forward Checking

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.

V_1 assignments

V_2 assignments

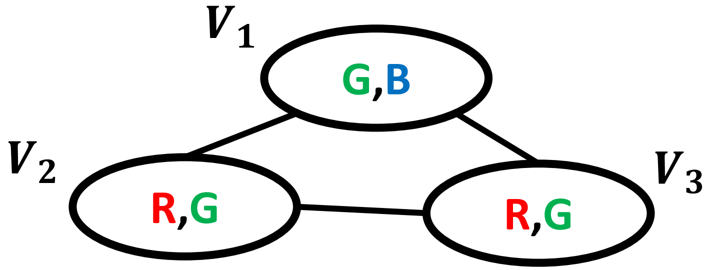
V_3 assignments



3. We have a conflict whenever a domain becomes empty.

→ Backtrack

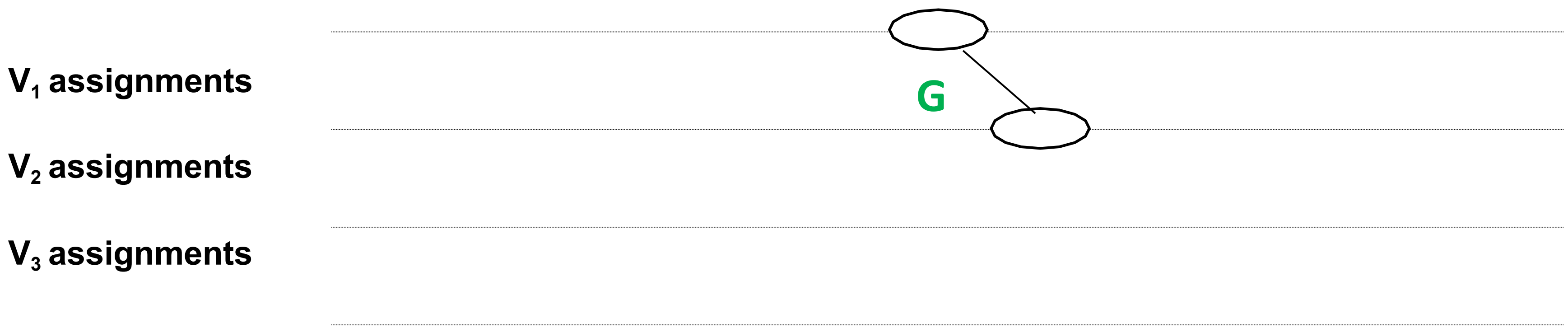
→ Restore Domains



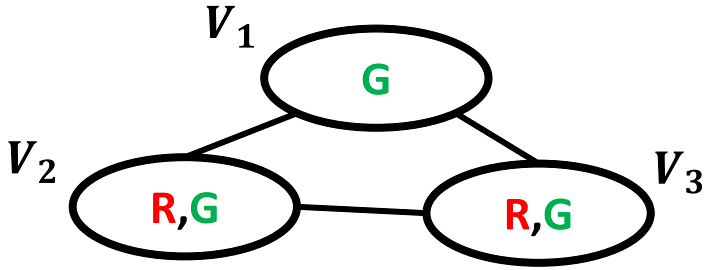
1. Perform initial pruning.

Backtracking with Forward Checking

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.



3. We have a conflict whenever a domain becomes empty.
   → Backtrack
   → Restore Domains



1. Perform initial pruning.

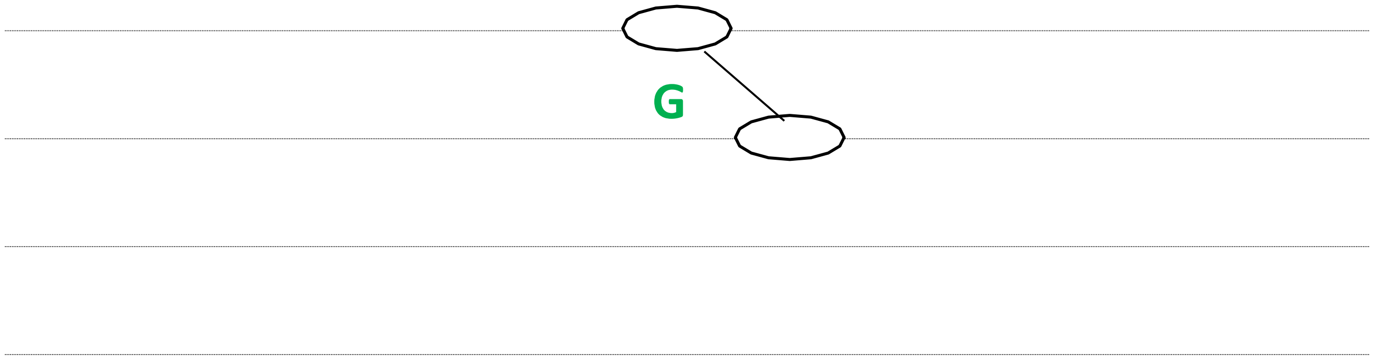
Backtracking with Forward Checking

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.

V_1 assignments

V_2 assignments

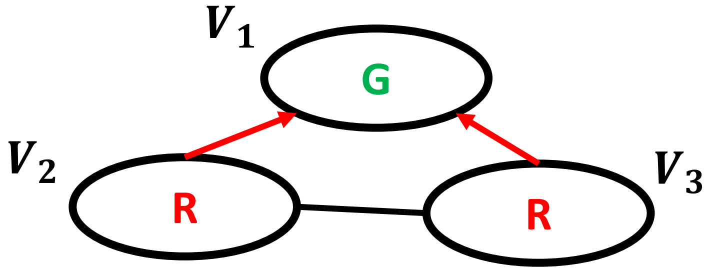
V_3 assignments



3. We have a conflict whenever a domain becomes empty.

→ Backtrack

→ Restore Domains

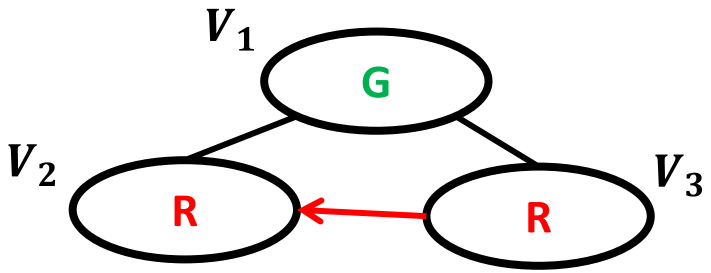


1. Perform initial pruning.

Backtracking with Forward Checking

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.

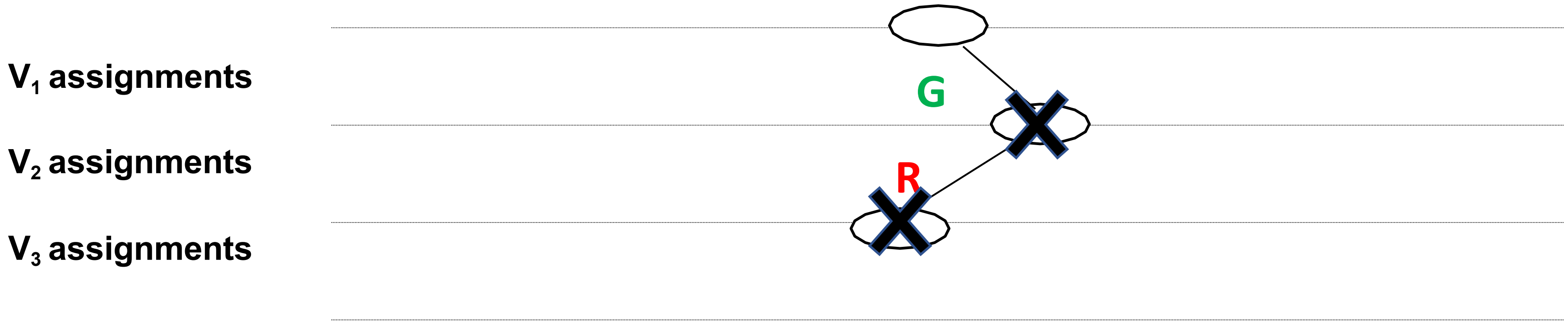
3. We have a conflict whenever a domain becomes empty.
   → Backtrack
   → Restore Domains



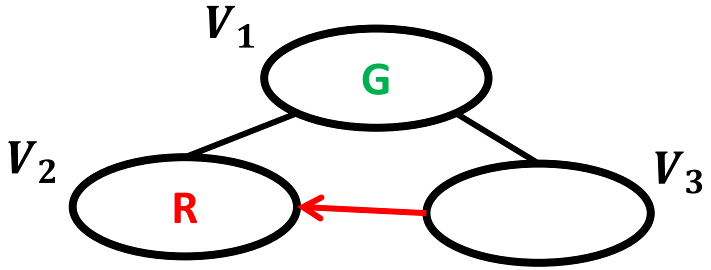
1. Perform initial pruning.

Backtracking with Forward Checking

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.



3. We have a conflict whenever a domain becomes empty.
   → Backtrack
   → Restore Domains



1. Perform initial pruning.

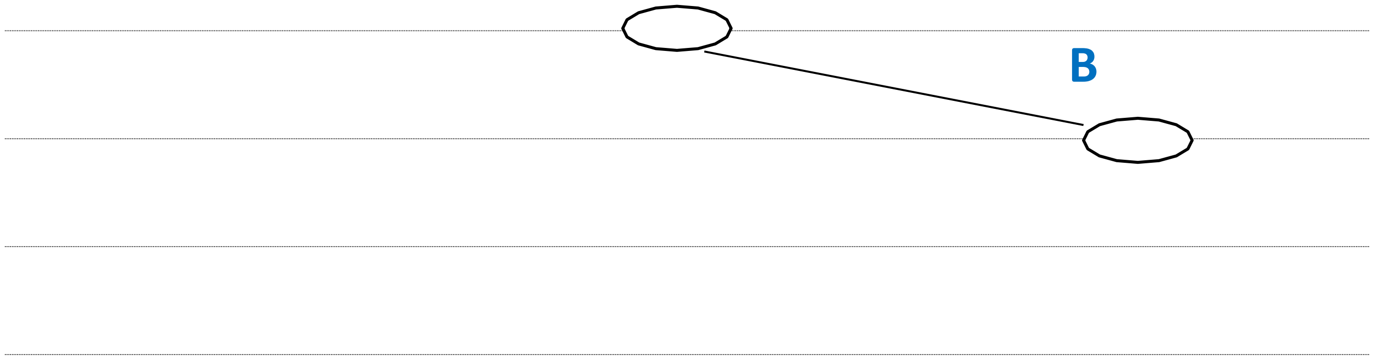
Backtracking with Forward Checking

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.

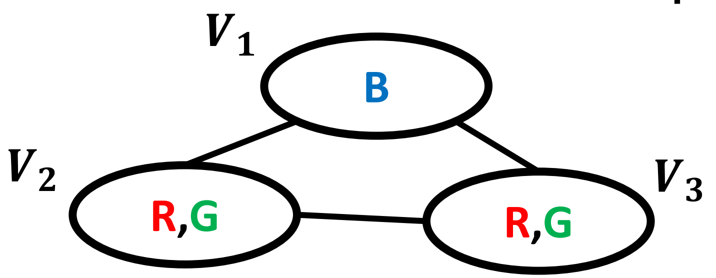
V_1 assignments

V_2 assignments

V_3 assignments



3. We have a conflict whenever a domain becomes empty.
   → Backtrack
   → Restore Domains



1. Perform initial pruning.

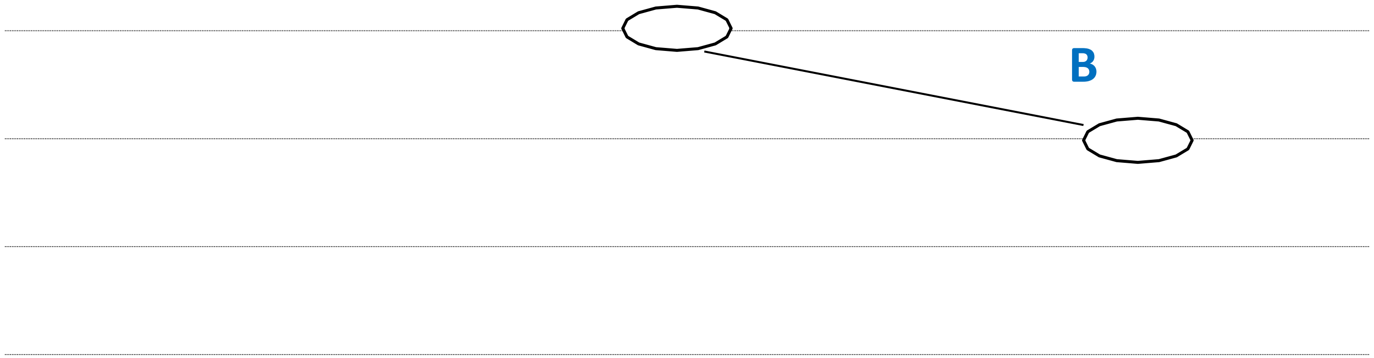
Backtracking with Forward Checking

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.

V_1 assignments

V_2 assignments

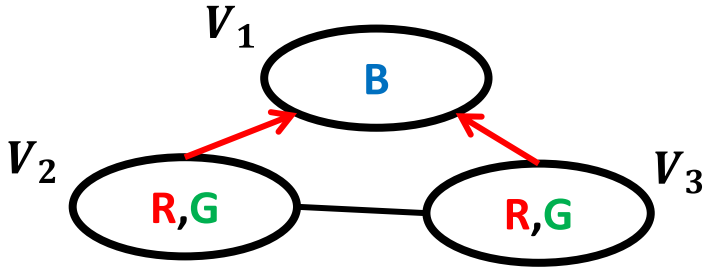
V_3 assignments



3. We have a conflict whenever a domain becomes empty.

→ Backtrack

→ Restore Domains



1. Perform initial pruning.

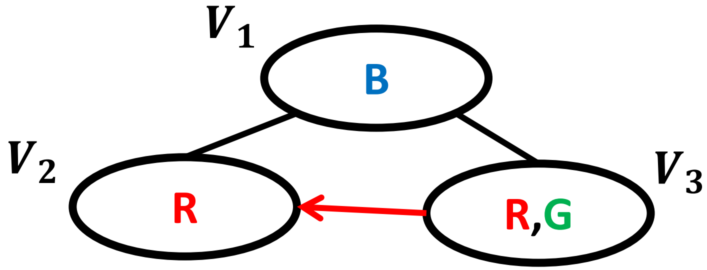
Backtracking with Forward Checking

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.

3. We have a conflict whenever a domain becomes empty.

→ Backtrack

→ Restore Domains



1. Perform initial pruning.

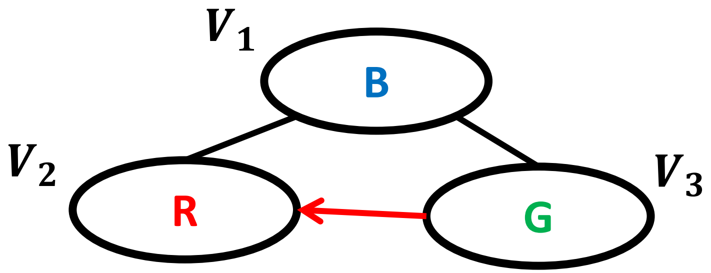
Backtracking with Forward Checking

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.

3. We have a conflict whenever a domain becomes empty.

→ Backtrack

→ Restore Domains



1. Perform initial pruning.

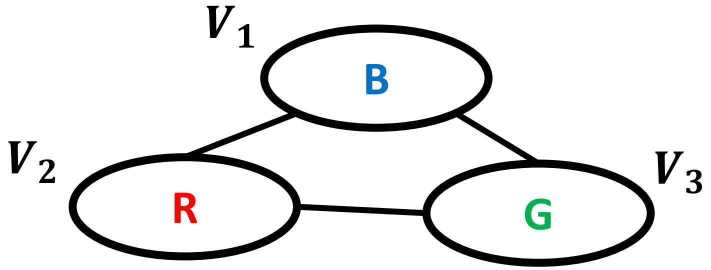
Backtracking with Forward Checking

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.

3. We have a conflict whenever a domain becomes empty.

→ Backtrack

→ Restore Domains



Solution!

1. Perform initial pruning.

Backtracking with Forward Checking

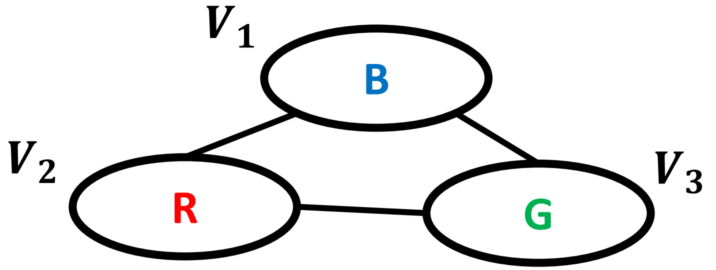
2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.

3. We have a conflict whenever a domain becomes empty.

→ Backtrack

→ Restore Domains

1. Perform initial pruning.



BT-FC is generally faster than pure BT because it avoids rediscovering inconsistencies.

Procedure Backtrack-Forward-Checking(x, D, C)

Input: A constraint network R = <X, D, C>
Output: A solution, or notification the network is inconsistent.
Note: Maintains n domain copies D’ for resetting, one for each search level i.

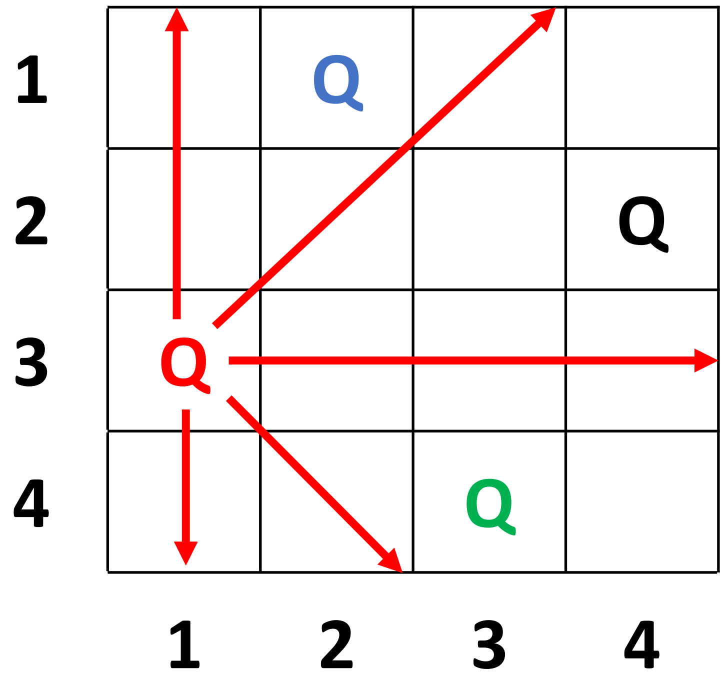
$D_i' \leftarrow D_i, \forall 1 \leq i \leq n$
$i \leftarrow 1; a_i = \{\ \ \}$
WHILE $1 \leq i \leq n$
    instantiate x_i ←Select-Value-FC()
    IF x_i = null
        reset each $D_k' | k \in \{i+1, \ldots, n\}$ to value before a was selected
        $i \leftarrow i - 1$
    ELSE
        $i \leftarrow i + 1$
ENDWHILE
IF $i = 0$
    RETURN “inconsistent”
ELSE
    RETURN $\vec{a}_i$, the instantiated values of $\{x_i, x_{i+1}, \ldots, x_n\}$

Procedure Select-Value-FC()

Output: A value in D_i' consistent with $\vec{a}_{i-1}$ or null if none. $O(ek^2)$

WHILE $D_i' \neq \emptyset$
 Pop $a \in D_i'$
 FOR all $k \in \{i+1, \ldots, n\}$
  FOR all $b \in D_k'$
   IF NOT(consistent($\vec{a}_{i-1}, x_i = a, x_k = b$))
    Remove b from D_k'
   ENDIF
  ENDFOR
 ENDFOR
 IF $\exists\ k \mid D_k' = \emptyset$
  reset each $D_k' | k \in \{i+1, \ldots, n\}$ to value before a was selected
 ELSE
  RETURN a
ENDWHILE
RETURN null

Search Performance on N Queens



- **Standard Search** // A handful of queens
- **Backtracking** // About 15 queens
- **BT with Forward Checking** // About 30 queens
- **Dynamic Variable Ordering**

Mid Lecture Break

To solve CSPs, we combine

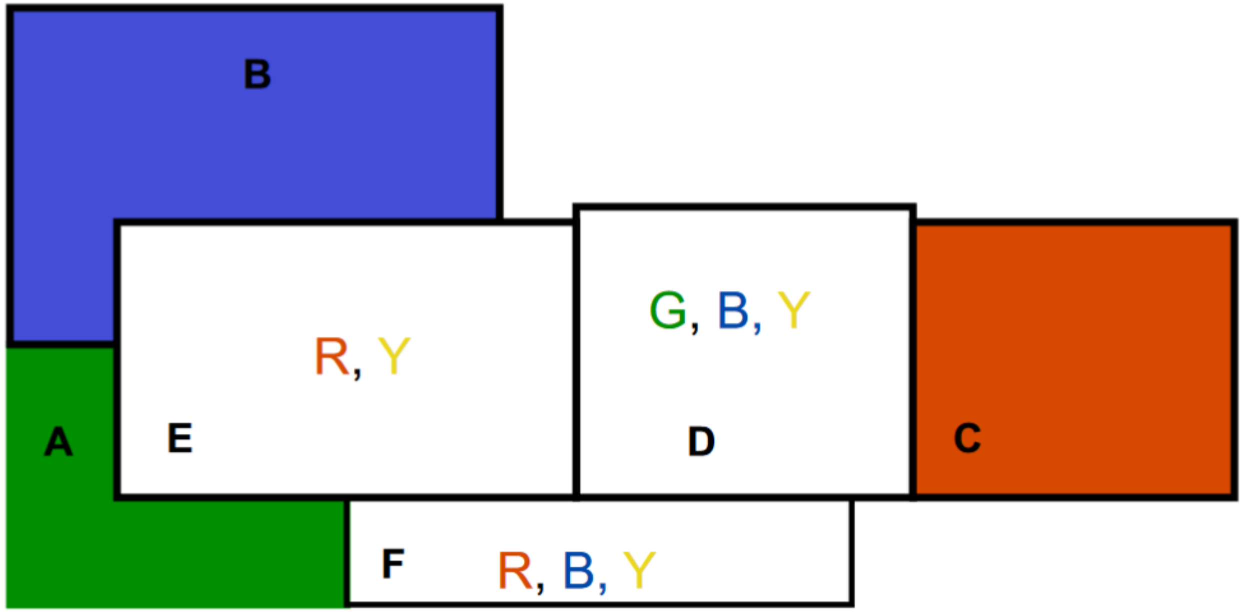
1. Arc consistency (constraint propagation),
    - Eliminates values that are shown locally to not be a part of any solution
2. Search
    - Explores consequences of committing to particular assignments
    - Methods incorporating search:
        - Standard Search
        - Backtrack Search (BT)
        - BT with Forward Checking (FC)
        - Dynamic Variable Ordering (DVO)
        - Iterative Repair
        - Back jumping (BJ)

BT-FC with dynamic ordering

- Traditional backtracking uses a fixed ordering over variables & values.
- Typically better to choose ordering dynamically as search proceeds.
- Most Constrained Variable
  - When doing forward-checking, pick variable with fewest legal values in domain to assign next → Minimizes branching factor.
- Least Constraining Value
  - Choose value that rules out the smallest number of values in variables connected to the chosen variable by constraints → Leaves most options to finding a satisfying assignment.

Example

Colors: R, G, B, Y



Which country should we color next? E most-constrained variable (smallest domain).

What color should we pick for it? RED least-constraining value (eliminates fewest values from neighboring domains).

Procedure Dynamic-Var-Forward-Checking(x,D,C)

Input: A constraint network R = <X, D, C>

Output: A solution, or notification the network is inconsistent.

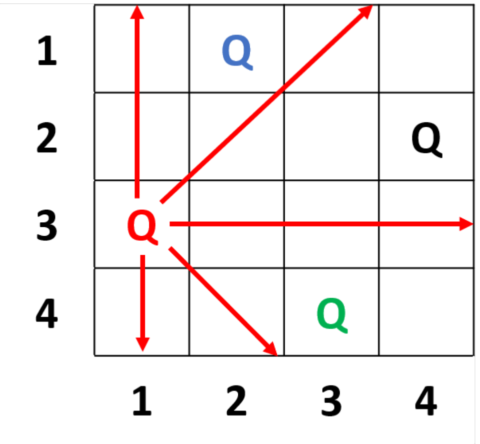
$D_i' \leftarrow D_i, \forall 1 \leq i \leq n$	Copy all domains
$i \leftarrow 1$; $a_i = \{\}$	Init variable counter and assignments
$s = \min_{i < j \leq n} \lvert D'_j \rvert$	Find unassigned variable w smallest domain
$x_{i+1} \leftarrow x_s$	Rearrange variables so that x_s follows x_i
while $1 \leq i \leq n$	
instantiate $x_i \leftarrow$ Select-Value-FC();	Select value (dynamic) and add to assignments, a_i
if x_i is null	No value to assign was returned.
reset each D'_k for $k > i$, to its value before x_i was last instantiated;	
$i \leftarrow i - 1$;	Backtrack
else	
if $I < n$	
$i \leftarrow i + 1$;	Step forward to x_s
$s = \min_{i < j \leq n} \lvert D'_j \rvert$	Find unassignedvariable w smallest domain
$x_{i+1} \leftarrow x_s$	Rearrange variables so that x_s follows x_i
else	
$i \leftarrow i + 1$;	Step forward to x_s
end while	
if $i = 0$	
return "inconsistent"	
else	
return $\vec{a_i}$, the instantiated values of $\{x_i, \ldots, x_n\}$	
end procedure	

Constraint Processing,

by R. Dechter

pgs 137-140

Search Performance on N Queens



- **Standard Search** // A handful of queens
- **Backtracking** // About 15 queens
- **BT with Forward Checking** // About 30 queens
- **Dynamic Variable Ordering** **// About 1,000 queens**
- **Iterative Repair** **// About 10,000,000 queens**
- **Conflict-directed Back Jumping**

Iterative Repair (Min-Conflict Heuristic)

1. Initialize a candidate solution using a "greedy" heuristic.
   - Gets the candidate "near" a solution

2. Select a variable in a conflict and assign it a value that minimizes the number of conflicts (break ties randomly).

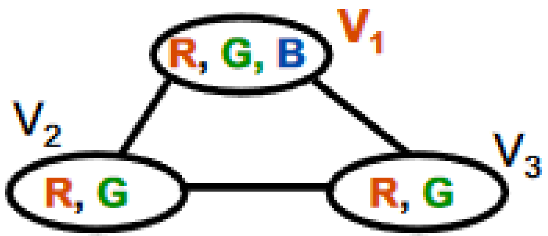
The heuristic is used in a local hill-climber (with or without backup)

Alternatives

Initial Assignment

 # conflicts

R R R: 3	BRR	GRR	RGR	RRG



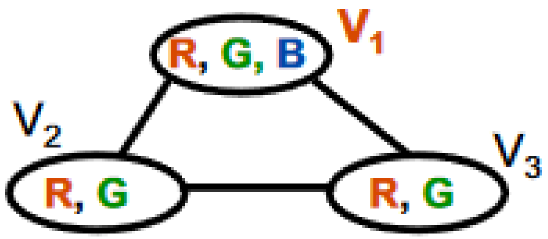
Iterative Repair (Min-Conflict Heuristic)

1. Initialize a candidate solution using a “greedy” heuristic.
   - Gets the candidate “near” a solution

2. Select a variable in a conflict and assign it a value that minimizes the number of conflicts (break ties randomly).

The heuristic is used in a local hill-climber (with or without backup)

R R R: 3	BRR	GRR	RGR	RRG
B R R: 1	RRR	GRR	BGR	BRG



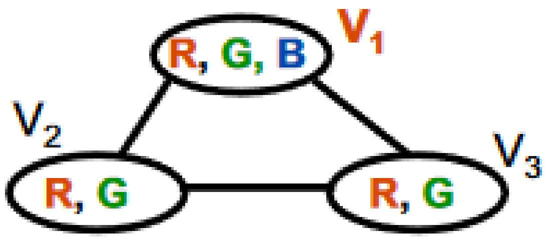
Iterative Repair (Min-Conflict Heuristic)

1. Initialize a candidate solution using a “greedy” heuristic.
   - Gets the candidate “near” a solution

2. Select a variable in a conflict and assign it a value that minimizes the number of conflicts (break ties randomly).

The heuristic is used in a local hill-climber (with or without backup)

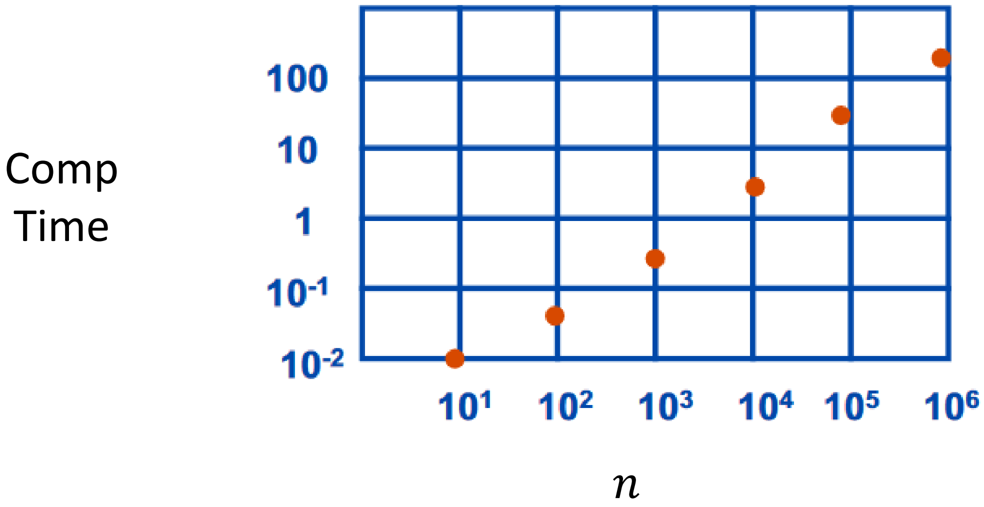
R R R: 3	BRR	GRR	RGR	RRG
B R R: 1	BRR	GRR	BGR	BRG
B G R: 0				



Min-Conflict Heuristic

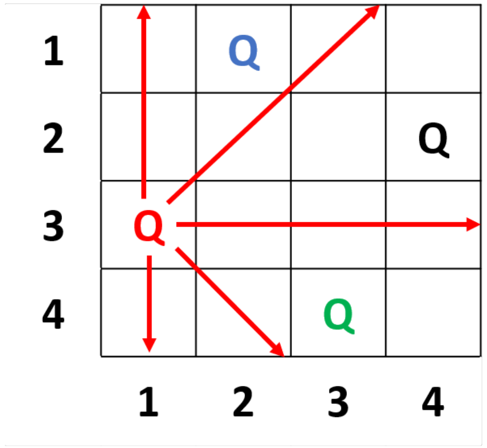
Pure hill climber (without backtracking) gets stuck in local minima:

- Add random moves to attempt to get out of local minima
- Add weights on violated constraints and increase weight every cycle the constraints remains violated



GSAT: Randomized hill climber used to solve propositional logic SATisfiability problems

Search Performance on N Queens



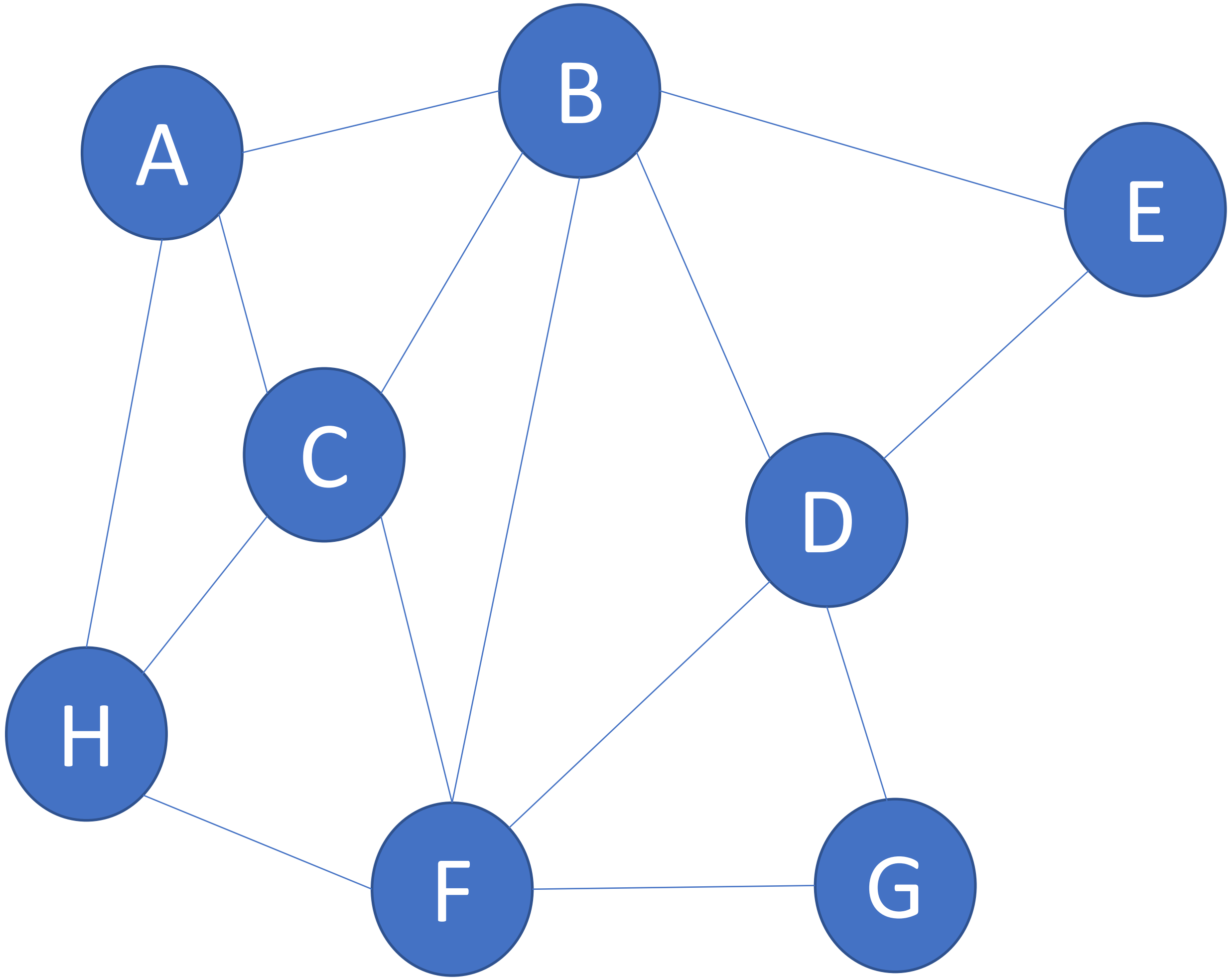
- **Standard Search** // A handful of queens
- **Backtracking** // About 15 queens
- **BT with Forward Checking** // About 30 queens
- **Dynamic Variable Ordering** **// About 1,000 queens**
- **Iterative Repair** **// About 10,000,000 queens**
- **Conflict-directed Back Jumping**

Back Jumping

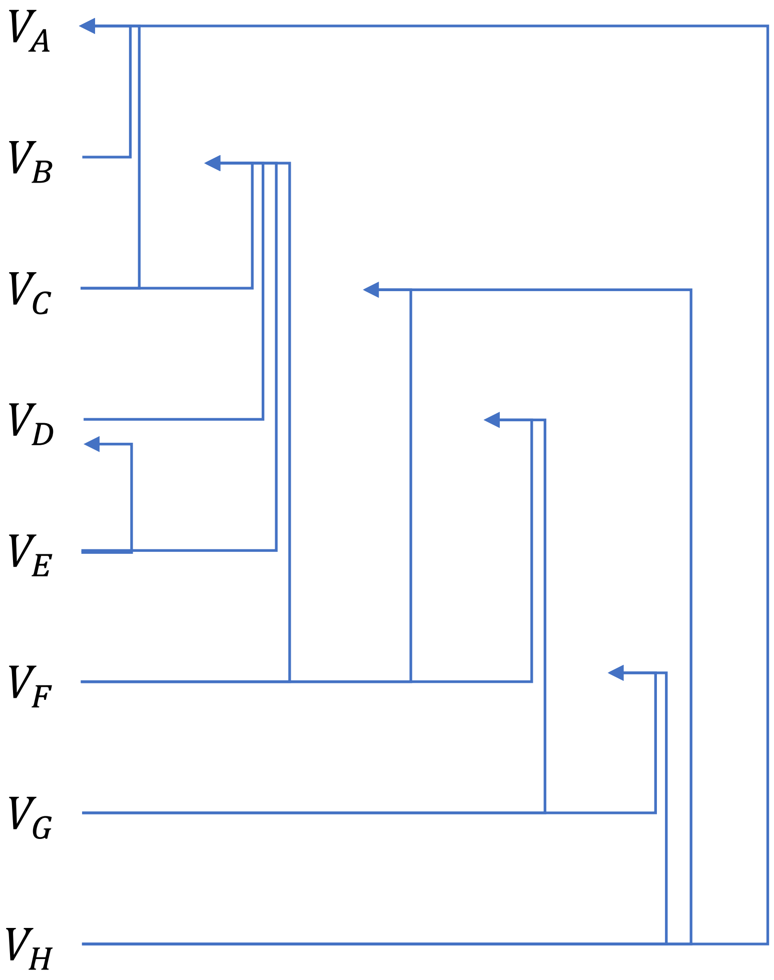
- **Backtracking:** At dead end, backup to the most recent variable.

- **Backjumping:** At dead end, backup to the most recent variable that eliminated some value in the domain of the dead-end variable.

Back Jumping

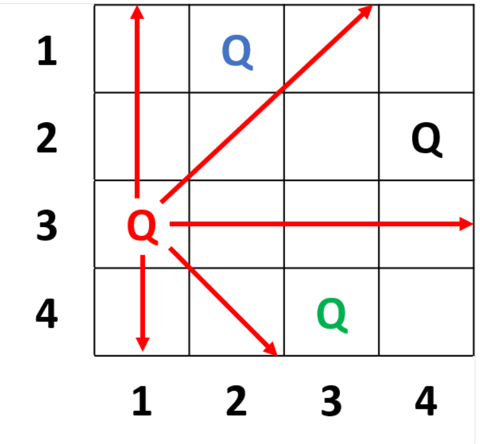


Variables and Instantiation order
Checking back



Slides from Prosser [4C presentation, 2003]

Search Performance on N Queens



- **Standard Search** // A handful of queens
- **Backtracking** // About 15 queens
- **BT with Forward Checking** // About 30 queens
- **Dynamic Variable Ordering** // About 1,000 queens
- **Iterative Repair** // About 10,000,000 queens
- **Conflict-directed Back Jumping**

To solve CSPs, we combine

1. Arc consistency (constraint propagation),
   - Eliminates values that are shown locally to not be a part of any solution
2. Search
   - Explores consequences of committing to particular assignments
   - Methods incorporating search:
     - Standard Search
     - Backtrack Search (BT)
     - BT with Forward Checking (FC)
     - Dynamic Variable Ordering (DVO)
     - Iterative Repair
     - Back jumping (BJ)