Solving Constraint Satisfaction Problems: Forward Checking

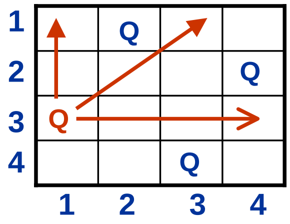
Brian C. Williams
16.410
October 1st, 2003

Slides adapted from:
6.034 Tomas Lozano Perez
With help from:
Stuart Russell & Peter Norvig

CSPS and Encoding 4 Queens

Problem: Place queens so that none can attack the other.

- **Assume one queen per column.**
- **What row should each queen be in?**



A Constraint Satisfaction Problem is a triple <V,D,C>:

Variables V $Q_1, Q_2, Q_3, Q_4,$

Domains D $\{1, 2, 3, 4\}$

Constraints C $Q_i <> Q_j$ On different rows

$|Q_i - Q_j| <> |i-j|$ Stay off the diagonals

Example: $C_{1,2} = \{(1,3)\ (1,4)\ (2,4)\ (3,1)\ (4,1)\ (4,2)\}$

CSP solution: any assignment to V, such that all constraints in C are satisfied.

Achieving Arc Consistency via Constraint Propagation

Arc consistency eliminates values of each variable domain that can never satisfy a particular constraint (an arc).

- Directed arc (V_i, V_j) is arc consistent if
  $\forall x \in D_i\ \exists y \in D_j$ such that (x,y) is allowed by constraint C_{ij}

$$\begin{array}{ccc} V_i & \rightarrow & V_j \\ \{1,2,\cancel{3}\} & = & \{1,2\} \end{array}$$

Constraint propagation: To achieve arc consistency:

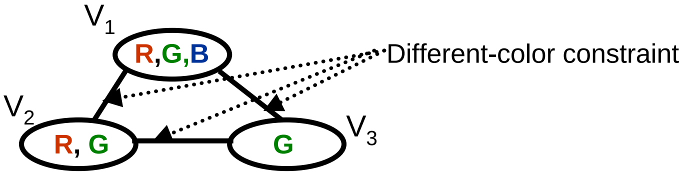
- Delete every value from each tail domain D_i of each arc that fails this condition.
  - Repeat until quiescence:
    - If element deleted from Di then

      check directed arc consistency for each arc with head D_i
  - Maintain arcs to be checked on FIFO queue (no duplicates).

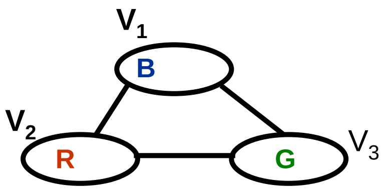
Constraint Propagation Example

Graph Coloring

Initial Domains



Arc examined	Value deleted
$V_1 – V_2$	none
V_1-V_3	V_1(G)
V_2 -V_3	V_2(G)
V_2-V_1	V_1(R)
V_2>V_1	**none**
V_3>V_1	**none**



Arcs to examine

IF **examination queue is empty**

THEN **arc (pairwise) consistent.**

To Solve CSPs we combine arc consistency and search

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
- Back Track search (BT)
- BT with Forward Checking (FC)
- Dynamic Variable Ordering (DV)
- Iterative Repair
- Backjumping (BJ)

Solving CSPs with Standard Search

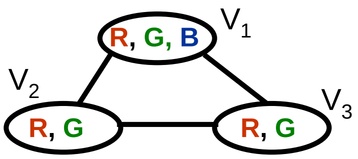
• State	• Variables assigned thus far
• Initial State	• No assignments
• Operator	• Assign value to any unassigned variable
• Goal Test	• All variables assigned • All constraints satisfied

- Branching factor?

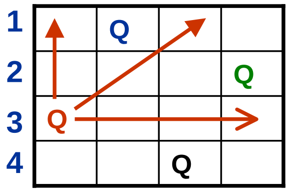
➔ **Sum of domain size of all variables** O(v*d)

- Performance?

➔ **exponential in branching factor**



Search Performance on N Queens



- Standard Search
- Backtracking

- A handful of queens

Solving CSPs with Standard Search

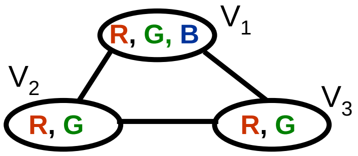
Standard Search:

- Children select any value to any variable [O(v*d)]
- Test complete assignments against CSP

Observations:

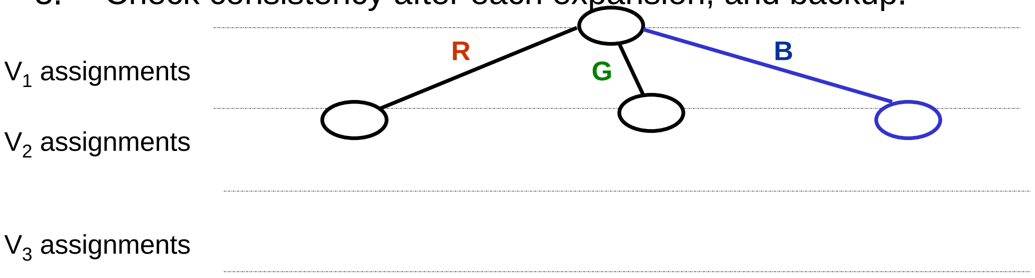
1. The order in which variables are assigned does not change the solution.
   ➔ **Many paths denote the same solution (n!),**
   ➔ **so expand only one path.**

2. We can identify a dead end before assigning all variables
   ➔ **Extensions to inconsistent partial assignments are always inconsistent**
   ➔ **So check after each assignment.**



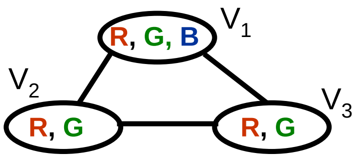
BackTrack Search (BT)

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



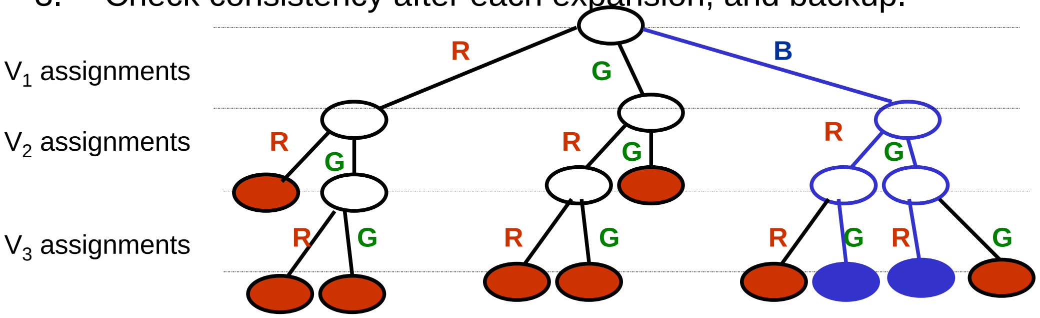
Select variable ordering to assign

Expand designated variable



BackTrack Search (BT)

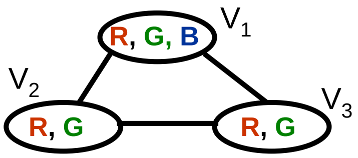
1. Expand the assignments of only one variable at each step.
2. Pursue depth first.
3. Check consistency after each expansion, and backup.



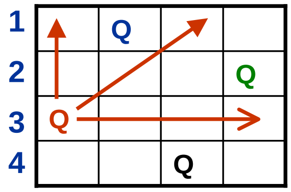
Select variable ordering to assign

Assign designated variable

Backup at inconsistent assignment



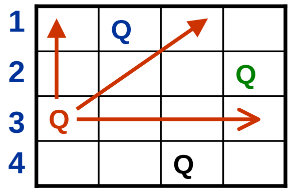
Search Performance on N Queens



- Standard Search
- Backtracking

- A handful of queens
- About 15 queens

Search Performance on N Queens



- Standard Search
- Backtracking
- BT with Forward Checking

- A handful of queens
- About 15 queens

Combine Backtracking & Limited Constraint Propagation

Initially: Prune domains using constraint propagation

Loop:

- If complete consistent assignment, then return.
- Choose unassigned variable
- Choose assignment from pruned domain
- Prune domains using constraint propagation
- if a domain has no remaining elements, then backtrack.

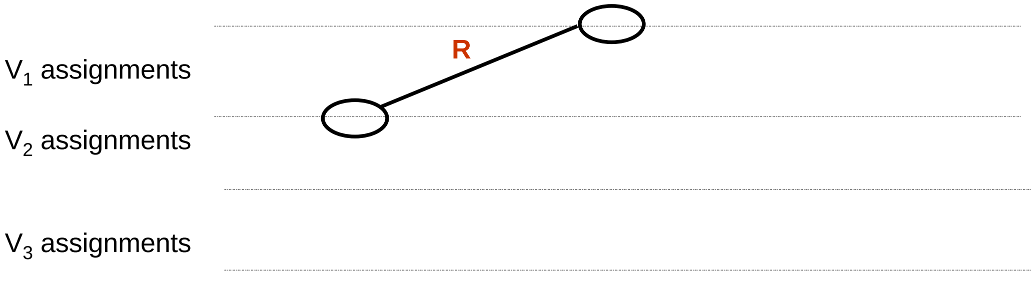
Question: Full propagation is $O(ed^3)$,
How much propagation should we do?

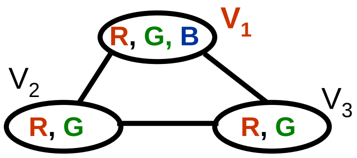
Very little:

- Just check arc consistency for those arcs terminating on the new assignment $O(ed)$.
- called **forward checking** (FC).

Backtracking with Forward Checking (BT-FC)

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

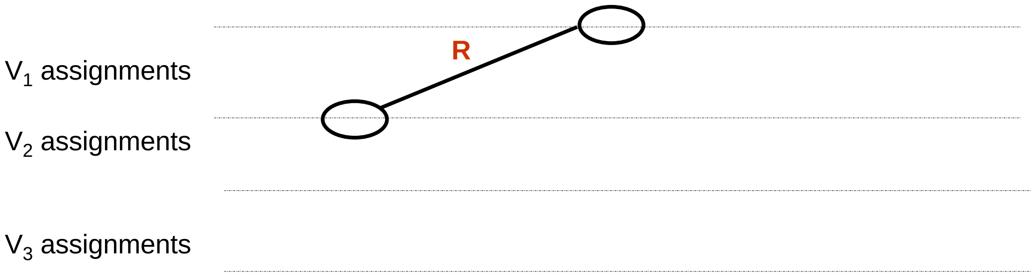




1. Perform initial pruning.

Backtracking with Forward Checking (BT-FC)

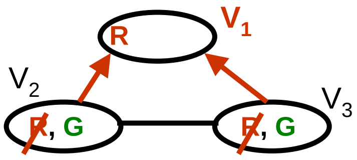
2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.



V_1 assignments

V_2 assignments

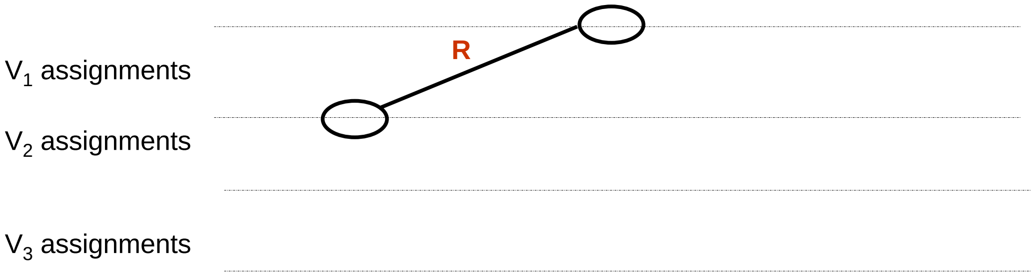
V_3 assignments



1. Perform initial pruning.

Backtracking with Forward Checking (BT-FC)

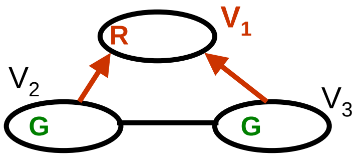
2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.



V_1 assignments

V_2 assignments

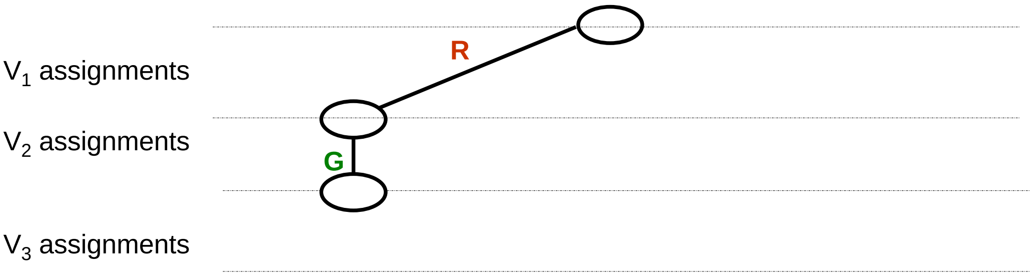
V_3 assignments

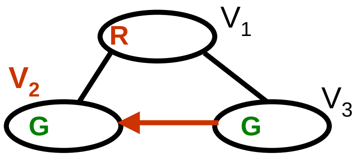


1. Perform initial pruning.

Backtracking with Forward Checking (BT-FC)

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

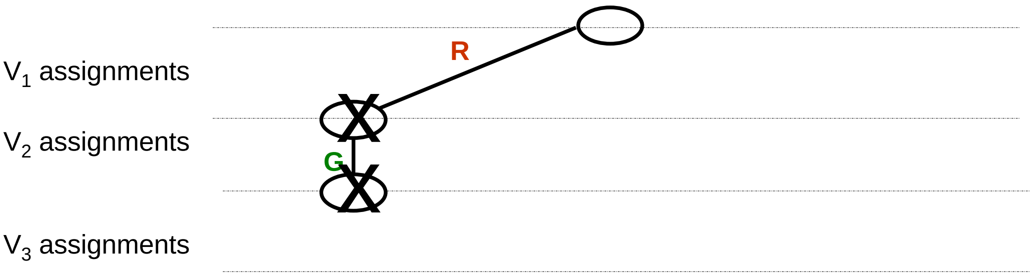




1. Perform initial pruning.

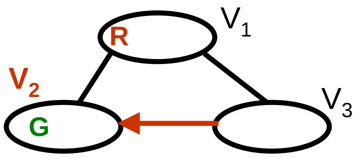
Backtracking with Forward Checking (BT-FC)

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.

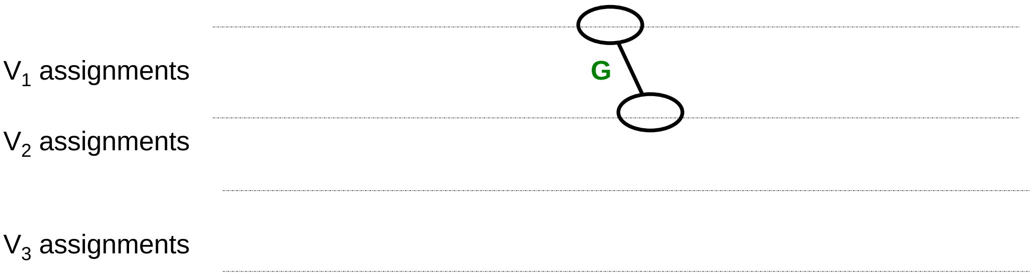
- Back track



1. Perform initial pruning.

Backtracking with Forward Checking (BT-FC)

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



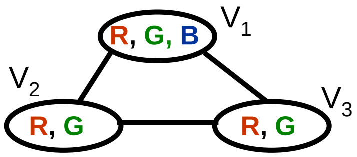
V_1 assignments

V_2 assignments

V_3 assignments

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

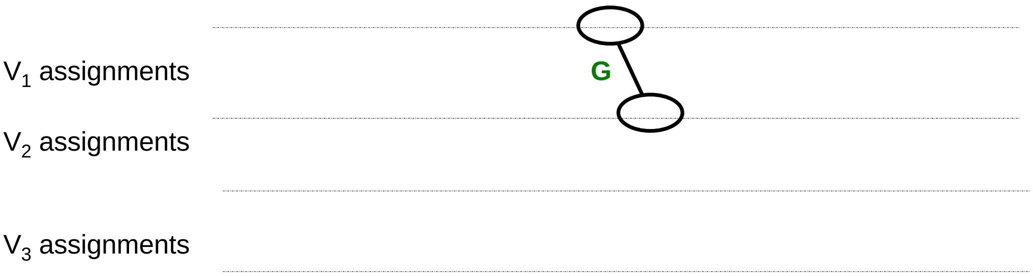
- Back track
- Restore domain values



1. Perform initial pruning.

Backtracking with Forward Checking (BT-FC)

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



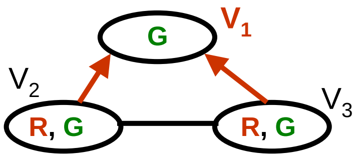
V_1 assignments

V_2 assignments

V_3 assignments

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

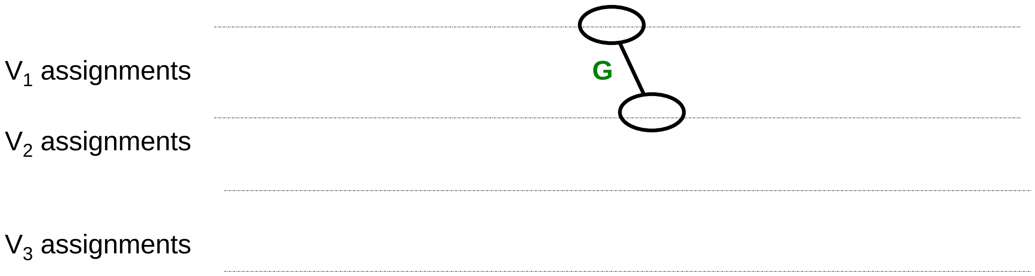
- Back track
- Restore domain values



1. Perform initial pruning.

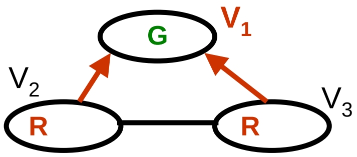
Backtracking with Forward Checking (BT-FC)

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.

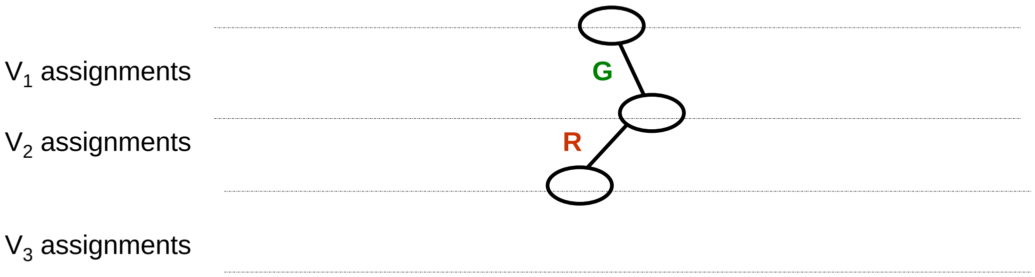
- Back track
- Restore domain values



1. Perform initial pruning.

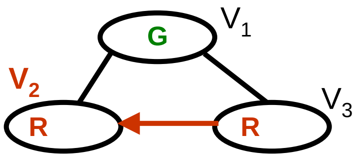
Backtracking with Forward Checking (BT-FC)

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.

- Back track
- Restore domain values

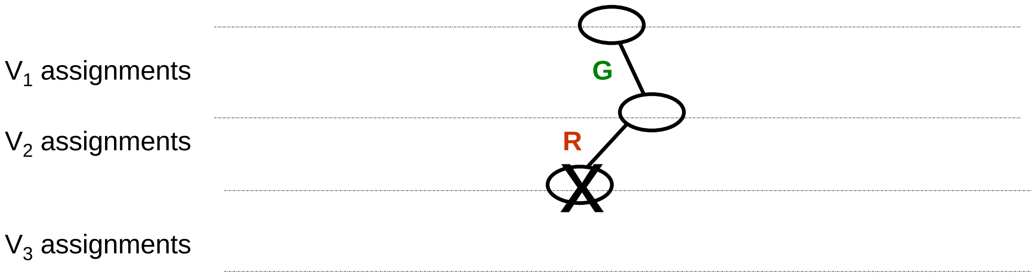


Note: No need to check new assignment against previous assignments

1. Perform initial pruning.

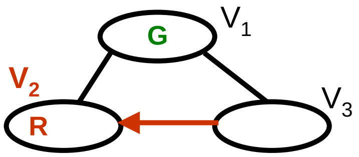
Backtracking with Forward Checking (BT-FC)

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.

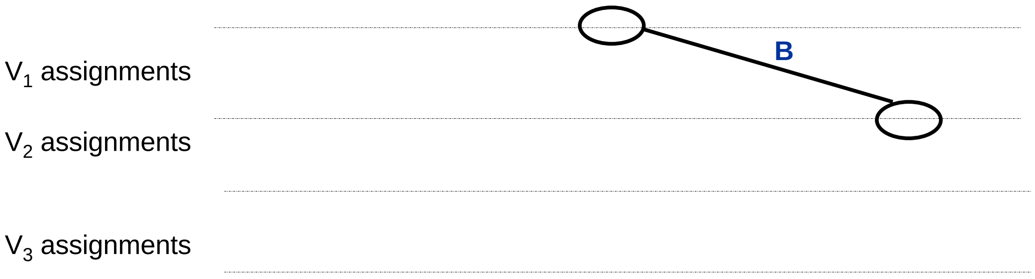
- Back track
- Restore domain values



1. Perform initial pruning.

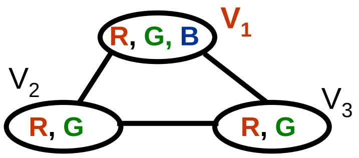
Backtracking with Forward Checking (BT-FC)

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.

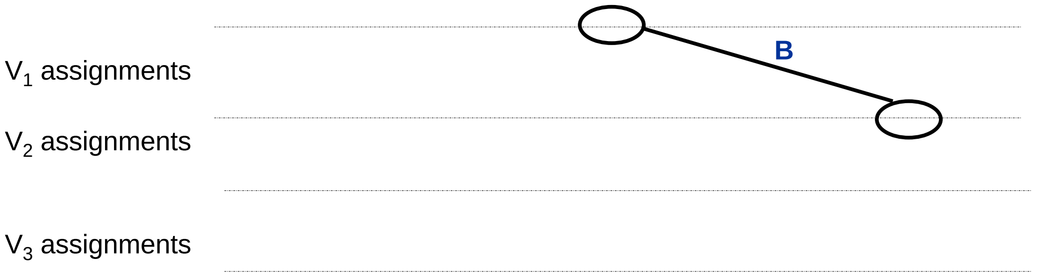
- Back track
- Restore domain values



1. Perform initial pruning.

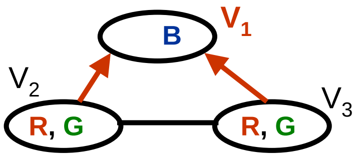
Backtracking with Forward Checking (BT-FC)

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.

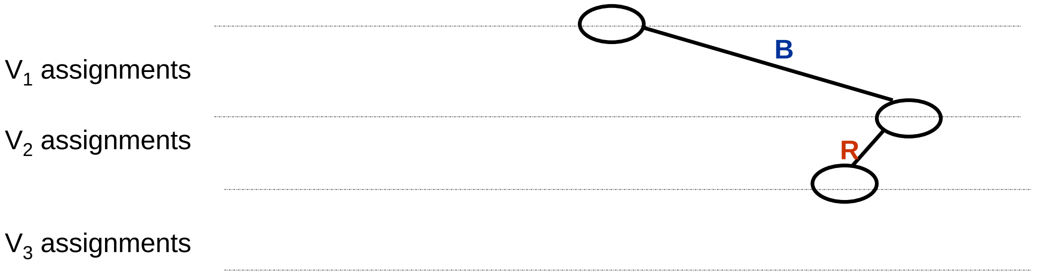
- Back track
- Restore domain values



1. Perform initial pruning.

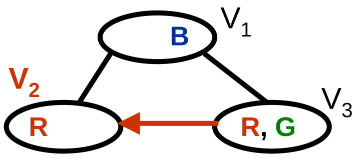
Backtracking with Forward Checking (BT-FC)

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.

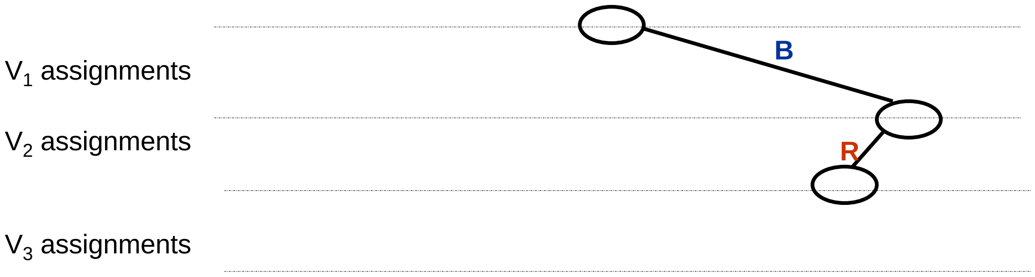
- Back track
- Restore domain values



1. Perform initial pruning.

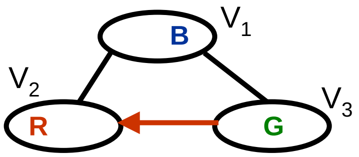
Backtracking with Forward Checking (BT-FC)

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.

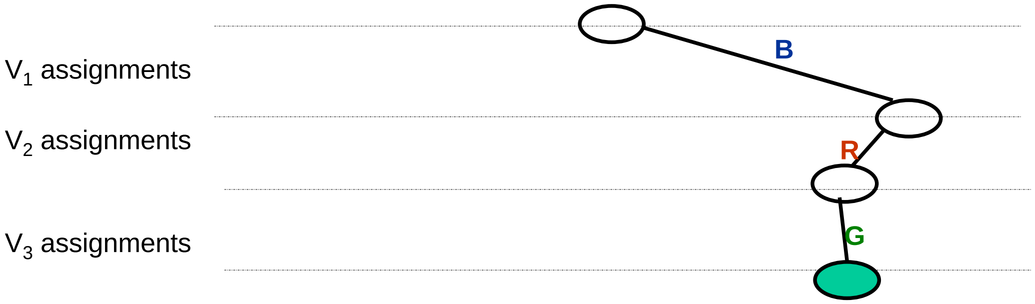
- Back track
- Restore domain values



1. Perform initial pruning.

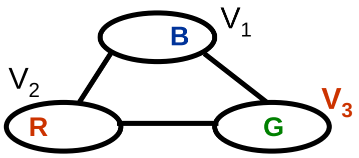
Backtracking with Forward Checking (BT-FC)

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.

- Back track
- Restore domain values

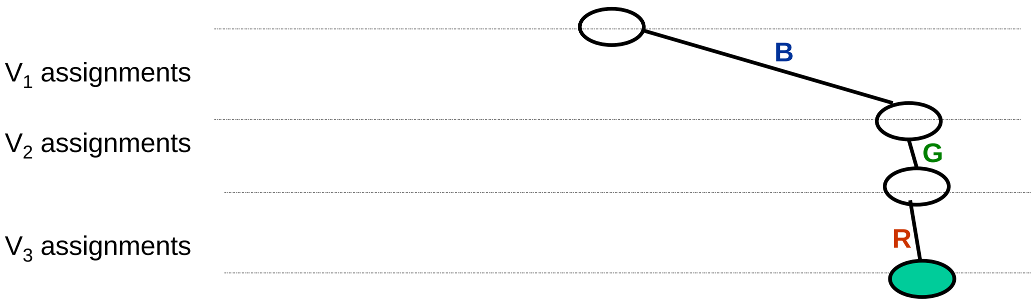


Solution!

1. Perform initial pruning.

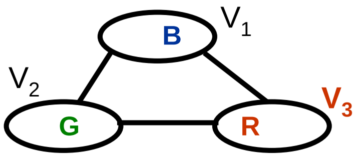
Backtracking with Forward Checking (BT-FC)

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.

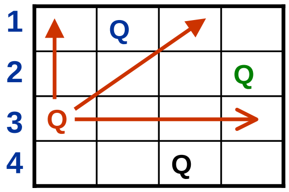
- Back track
- Restore domain values



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

1. Perform initial pruning.

Search Performance on N Queens



- Standard Search
- **Backtracking**
- **BT with Forward Checking**

- A handful of queens
- About 15 queens
- About 30 queens

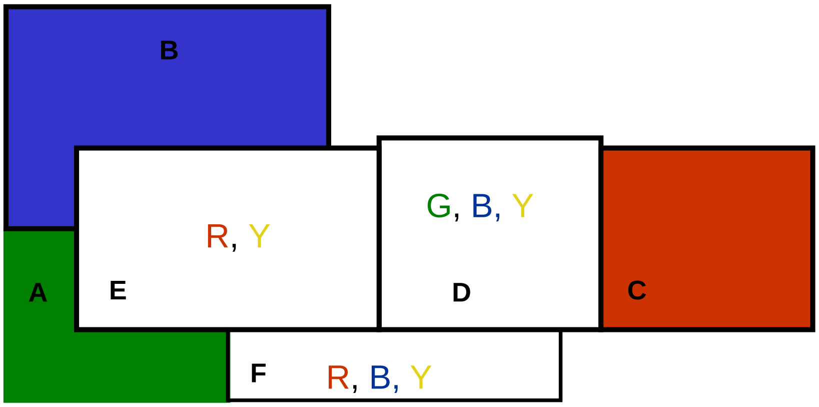
BT-FC with dynamic ordering

Traditional backtracking uses fixed ordering of variables & values

Typically better to choose ordering dynamically as search proceeds.

- Most constrained variable
  when doing forward-checking, pick variable with fewest legal values 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.

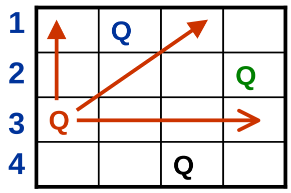
Colors: R, G, B, Y



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)

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

Back jumping

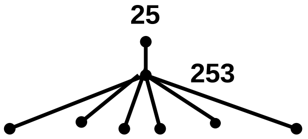
Backtracking At dead end backup to most recent variable,

Backjumping At dead end backup to most recent variable that eliminated a value in the current (empty) domain.

1		1			3	2
2	Q	1	1	1	1	1
3		1	Q	2	3	3
4			1	3		
5		Q	2	1	2	2
6			2		1	3
	1	2	3	4	5	6

2
3
4
5
6

6-Queens
variables: board columns
domains: board rows



Back jumping

Backtracking At dead end backup to most recent variable,

Backjumping At dead end backup to most recent variable that eliminated a value in the current (empty) domain.

1		1			3	2
2	Q	1	1	1	1	1
3	4	1	Q	2	3	3
4		4	1	3		4
5		Q	2	1	2	2
6			2	Q	1	3
	1	2	3	4	5	6

2

3

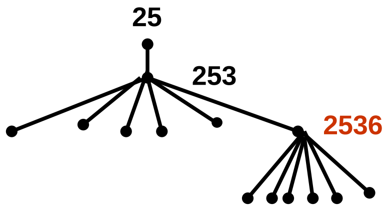
4

5

6

6-Queens

variables: board columns
domains: board rows



Back jumping

Backtracking At dead end backup to most recent variable,

Backjumping At dead end backup to most recent variable that eliminated a value in the current (empty) domain.

1		1			3	2
2	Q	1	1	1	1	1
3	4	1	Q	2	3	3
4		4	1	3	Q	4
5		Q	2	1	2	2
6			2	Q	1	3
	1	2	3	4	5	6

2

3

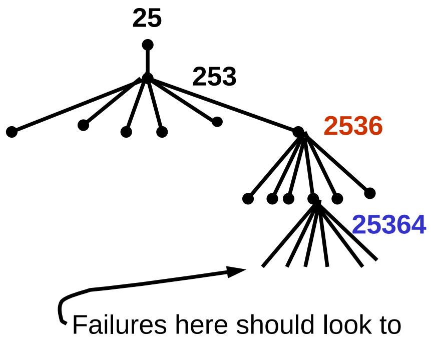
4

5

6

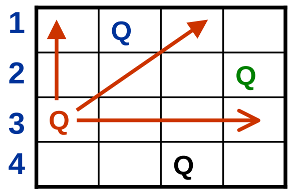
6-Queens

variables: board columns
domains: board rows



Failures here should look to variable 4. Changing variable 5 won't help

Search Performance on N Queens



Standard Search	A handful of queens
Backtracking	About 15 queens
Backjumping	???
BT with Forward Checking	About 30 queens
Dynamic Variable Ordering	About 1,000 queens
Iterative Repair	About 10,000,000 queens (except truly hard problems)