Solving Constraint Programs using Backtrack Search and Forward Checking

Slides draw upon material from:
6.034 notes, by Tomas Lozano Perez
AIMA, by Stuart Russell & Peter Norvig
Constraint Processing, by Rina Dechter

Brian C. Williams
16.410-13
September 27th, 2010

Assignments

- Remember:
  - Problem Set #3: Analysis and Constraint Programming, due this Wed., Sept. 29th, 2010.
- Reading:
  - Today: *[AIMA] Ch. 6.2-5*; Constraint Satisfaction.
  - Wednesday: Operator-based Planning *[AIMA]* Ch. 10 "Graph Plan," by Blum & Furst, posted on Stellar.
- To Learn More: *Constraint Processing*, by Rina Dechter
  - Ch. 5: General Search Strategies: Look-Ahead
  - Ch. 6: General Search Strategies: Look-Back
  - Ch. 7: Stochastic Greedy Local Search

Constraint Problems are Everywhere

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

(a) Sudoku Puzzle

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

(b) The Solution

Constraint Satisfaction Problems (CSP)

Input: A Constraint Satisfaction Problem is a triple <V,D,C>, 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 a set of constraints on assignments to V
  - Each constraint C_i = <S_i,R_i> specifies allowed variable assignments.
  - S_i the constraint's scope, is a subset of variables V.
  - R_i the constraint's relation, is a set of assignments to S_i.

Output: A full assignment to V, from elements of V's domain, such that all constraints in C are satisfied.

Constraint Modeling (Programming) Languages

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

Example: Constraint Model of the Sudoku Puzzle in Number Jack (http://4c110.ucc.ie/numberjack/home)

```
matrix = Matrix(N*N,N*N,1,N*N)
sudoku = Model( [AllDiff(row) for row in matrix.row],
                [AllDiff(col) for col in matrix.col],
                [AllDiff(matrix[x:x+N, y:y+N].flat)
                    for x in range(0,N*N,N)
                    for y in range(0,N*N,N)] )
```


Constraint Problems are Everywhere

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

(a) Sudoku Puzzle

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

(b) The Solution

Outline

- Analysis of constraint propagation
- Solving CSPs using Search

What is the Complexity of AC-1?

AC-1(CSP)

Input: A network of constraints CSP = <X, D, C>.

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

1. repeat

2. **for** every $c_{ij} \in C$,

3. Revise(x_i, x_j)

4. Revise(x_j, x_i)

5. **endfor**

6. until no domain is changed.

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.**
- **There are e binary constraints.**

Which is the correct complexity?

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

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)**
2. **if** there is no $a_j \in D_j$ such that $<a_i, a_j> \in R_{ij}$ — *** O(k)**
3. then delete a_i from D_i.
4. **endif**
5. **endfor**

Complexity of Revise?
= $O(k^2)$

where $k = \max_i |D_i|$

Full Arc-Consistency via AC-1

AC-1(CSP)

Input: A network of constraints CSP = <X, D, C>.

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

1. **repeat**
2. **for** every $c_{ij} \in C$, — **O(2e*revise)**
3. Revise(x_i, x_j)
4. Revise(x_j, x_i)
5. **endfor**
6. **until no domain is changed.** — *** O(nk)**

Complexity of AC-1?

= O(nk*e*revise)

= O(enk³)

where $k = \max_i |D_i|$

$n = |X|$, $e = |C|$

What is the Complexity of Constraint Propagation using AC-3?

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. **$O(ek^2)$**
3. **$O(ek^3)$**
4. **$O(ek)$**

Full Arc-Consistency via AC-3

AC-3(CSP)

Input: A network of constraints CSP = <X, D, C>.

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

```
for every c_ij ∈ C,                                              O(e) +
   queue ← queue ∪ {<x_i,x_j>, <x_i,x_j>}
endfor
while queue ≠ {}
   select and delete arc <x_i, x_j> from queue
      Revise(x_i, x_j)                                           O(k^2)
      if Revise(x_I, x_J) caused a change in D_i.                * O(ek)
         then queue ← queue ∪ {<x_k,x_I> | k ≠ i, k ≠ j}
      endif
endwhile
```

Complexity of AC-3?

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

where $k = \max_i |D_i|$, $n = |X|$, $e = |C|$

Is arc consistency sound and complete?

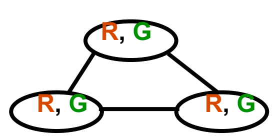
An *arc consistent solution* selects a value for every variable from its arc consistent domain.

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

- **Yes**
- **No**

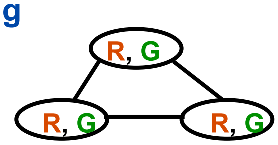
Completeness: All arc-consistent solutions are solutions to the CSP?

- **Yes**
- **No**

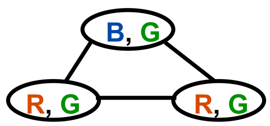


Incomplete: Arc consistency doesn't rule out all infeasible solutions

Graph Coloring



arc consistent, but <u>no</u> solutions.



arc consistent, but <u>2</u> solutions, not 8.

B,R,G
B,G,R

To Solve CSPs We Combine

1. Arc consistency (via 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 That Incorporate Search:

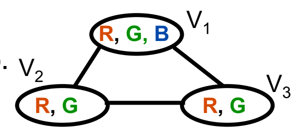
- Standard Search
- Back Track Search (BT)
- BT with Forward Checking (FC)
- Dynamic Variable Ordering (DV)
- Iterative Repair (IR)
- Conflict-directed Back Jumping (CBJ)

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** $O(|v|*|d|)$.

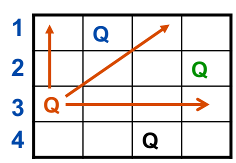


- Performance?

➔ **Exponential in the branching factor** $O([|v|*|d|]^{|v|})$.

17

Search Performance on N Queens



- Standard Search
- Backtracking
- A handful of queens

18

Solving CSPs with Standard Search

Standard Search:

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

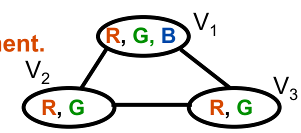
Observations:

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).**

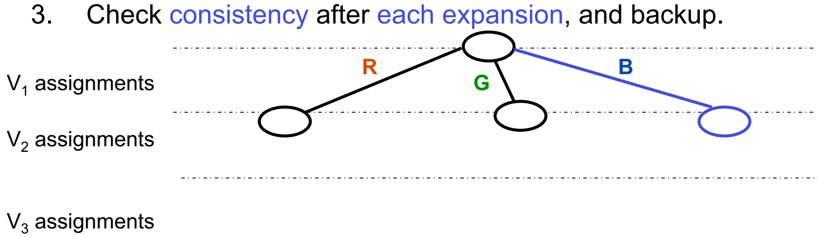
2. We can identify a dead end before we assign all variables.
   - **Extensions to inconsistent partial assignments are always inconsistent.**

   ➔ **Check consistency after each assignment.**



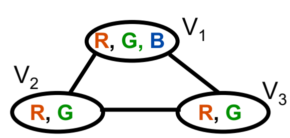
Back Track Search (BT)

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



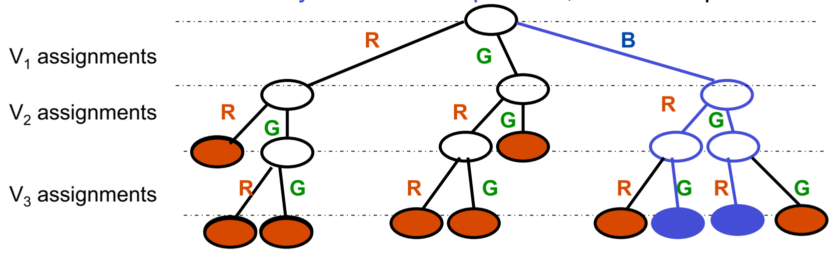
Preselect order of variables to assign

Assign designated variable



Back Track Search (BT)

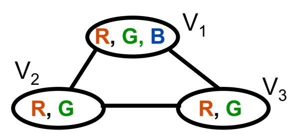
1. Expand assignments of one variable at each step.
2. Pursue depth first.
3. Check consistency after each expansion, and backup.



Preselect order of variables to assign

Assign designated variable

Backup at inconsistent assignment



21

Procedure Backtracking(<X,D,C>)

Input: A constraint network R = <X, D, C>
Output: A solution, or notification that the network is inconsistent.

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

22

Procedure Select-Value()

Output: A value in D'_i consistent with $\vec{a}_{i-1}$, or null, if none.

```
while D'_i is not empty
  select an arbitrary element a ∈ 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
```

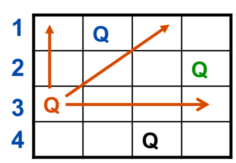
Constraint Processing,

by R. Dechter

pgs 123-127

23

Search Performance on N Queens



- Standard Search
- Backtracking
- BT with Forward Checking

- A handful of queens
- About 15 queens

24

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?

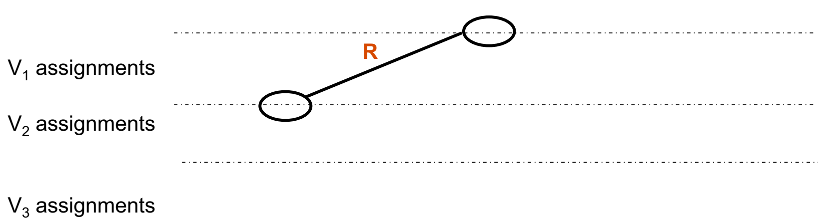
Very little (except for big problems)

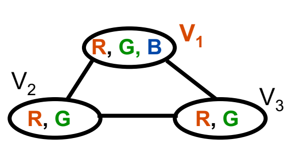
Forward Checking (FC)

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

Backtracking with Forward Checking (BT-FC)

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

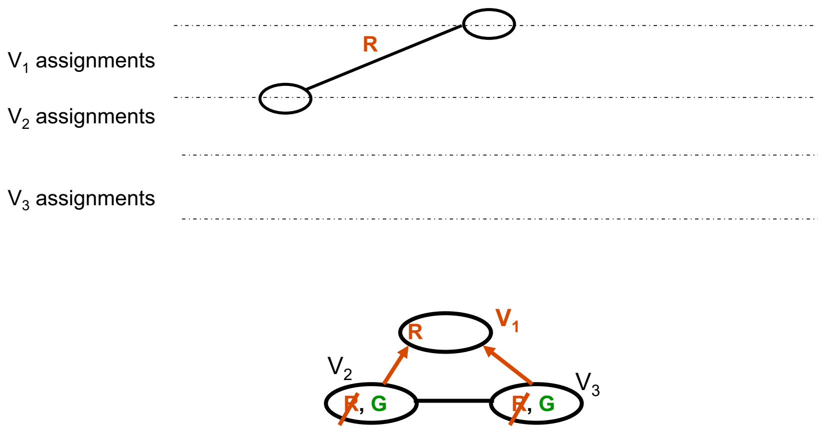




1. Perform initial pruning.

Backtracking with Forward Checking (BT-FC)

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

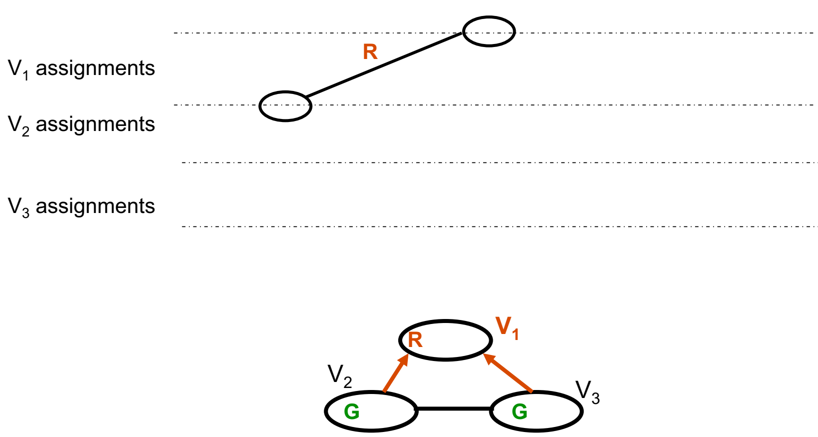


1. Perform initial pruning.

27

Backtracking with Forward Checking (BT-FC)

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

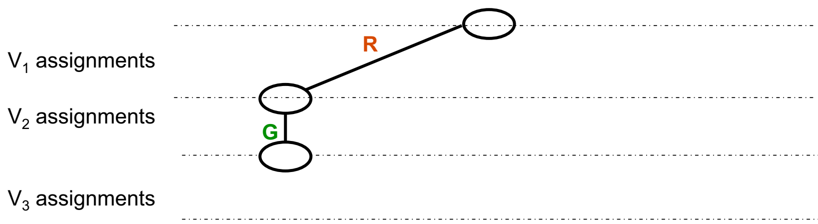


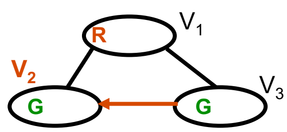
1. Perform initial pruning.

28

Backtracking with Forward Checking (BT-FC)

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



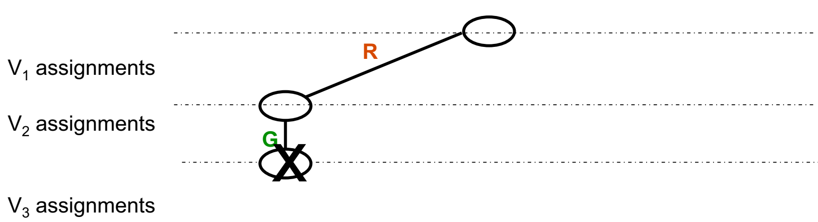


Note: No need to check new assignment against previous assignments

1. Perform initial pruning.

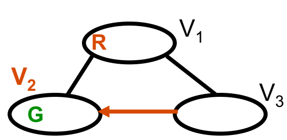
Backtracking with Forward Checking (BT-FC)

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.

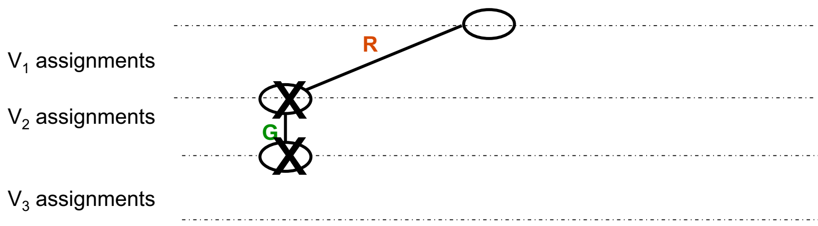
- Backtrack



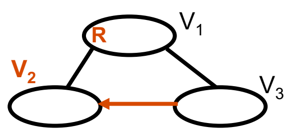
1. Perform initial pruning.

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.
   - Backtrack

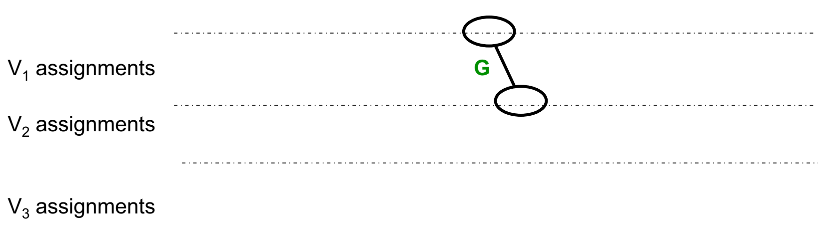


1. Perform initial pruning.

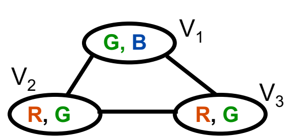
31

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.
   - Backtrack
   - Restore domains

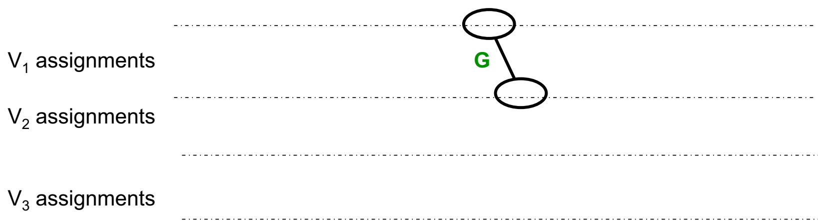


1. Perform initial pruning.

32

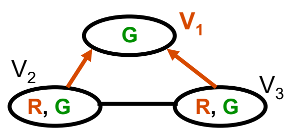
Backtracking with Forward Checking (BT-FC)

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.

- Backtrack
- Restore domains

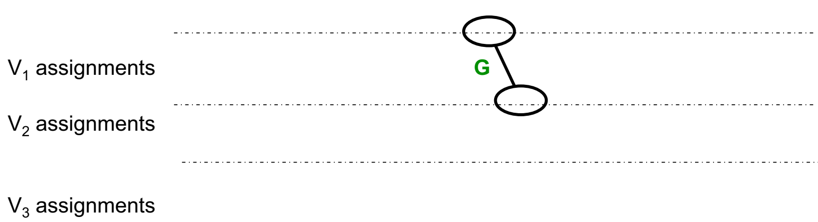


1. Perform initial pruning.

33

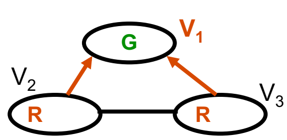
Backtracking with Forward Checking (BT-FC)

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.

- Backtrack
- Restore domains

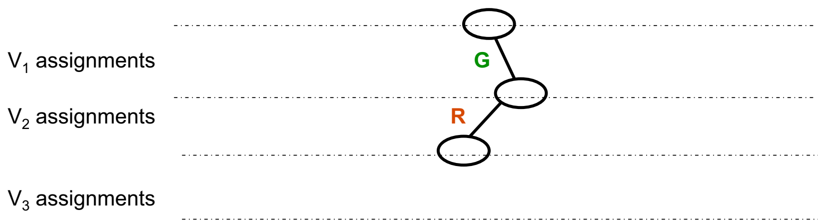


1. Perform initial pruning.

34

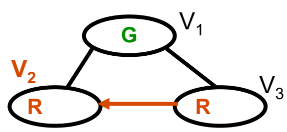
Backtracking with Forward Checking (BT-FC)

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.

- Backtrack
- Restore domains

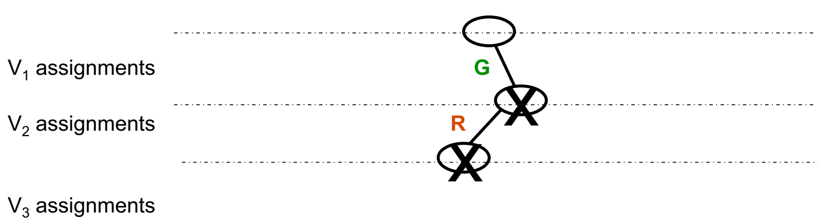


1. Perform initial pruning.

35

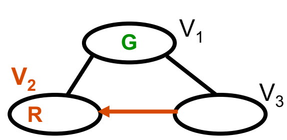
Backtracking with Forward Checking (BT-FC)

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.

- Backtrack
- Restore domains

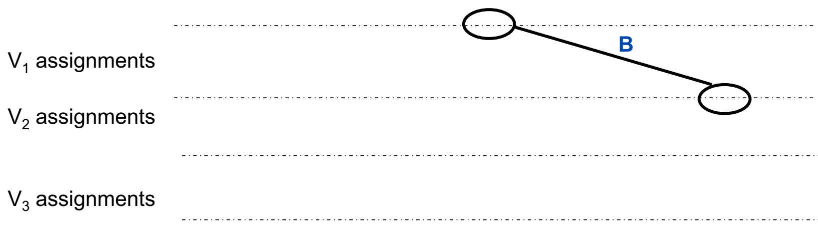


1. Perform initial pruning.

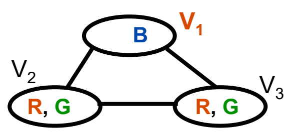
36

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.
   - Backtrack
   - Restore domains

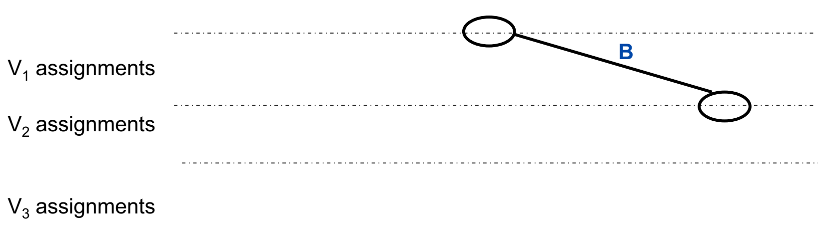


1. Perform initial pruning.

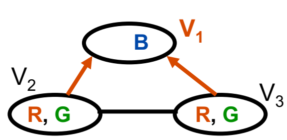
37

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.
   - Backtrack
   - Restore domains



1. Perform initial pruning.

38

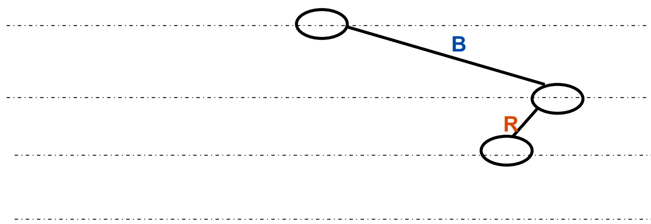
Backtracking with Forward Checking (BT-FC)

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

V_1 assignments

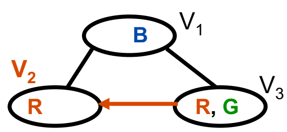
V_2 assignments

V_3 assignments



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

- Backtrack
- Restore domains



1. Perform initial pruning.

39

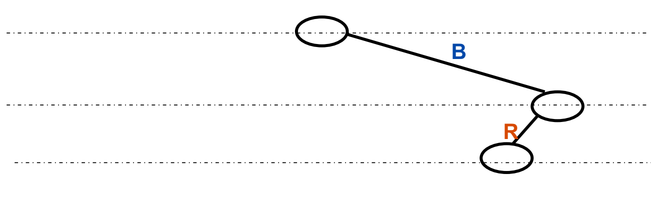
Backtracking with Forward Checking (BT-FC)

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

V_1 assignments

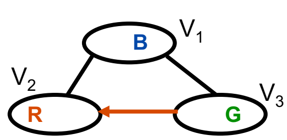
V_2 assignments

V_3 assignments



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

- Backtrack
- Restore domains

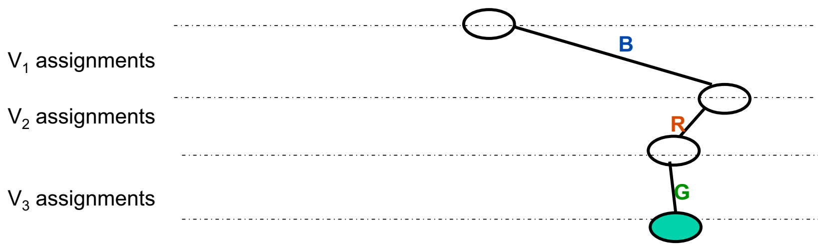


1. Perform initial pruning.

40

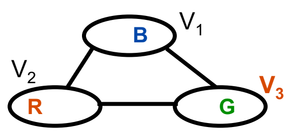
Backtracking with Forward Checking (BT-FC)

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.

- Backtrack
- Restore domains



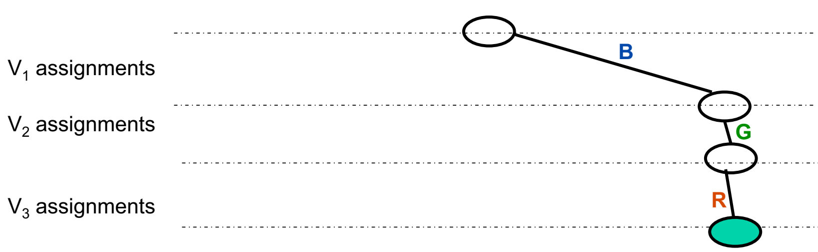
Solution!

1. Perform initial pruning.

41

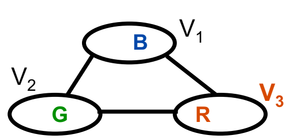
Backtracking with Forward Checking (BT-FC)

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.

- Backtrack
- Restore domains



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

1. Perform initial pruning.

42

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 ← D_i for 1 ≤ i ≤ n;                  (copy all domains)
i ← 1;  a_i = {}                           (init variable counter, assignments)
while 1 ≤ i ≤ n
  instantiate x_i ← Select-Value-FC();     (add to assignments, making a_i)
  if x_i is null                           (no value was returned)
    reset each D'_k for k > i, to its value before x_i was last instantiated;
    i ← i - 1;                             (backtrack)
  else
    i ← i + 1;                             (step forward)
end while
if i = 0
  return "inconsistent"
else
  return a_i , the instantiated values of {x_i, …, x_n}
end procedure
```

Constraint Processing,
by R. Dechter
pgs 131-4, 141

43

Procedure Select-Value-FC()

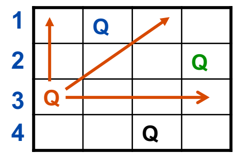
Output: A value in D'_i consistent with $\vec{a}_{i-1}$, or null, if none. $O(ek^2)$

```
while D'_i is not empty
  select an arbitrary element a ∈ D'_i and remove a from D'_i;
  for all k, i < k ≤ n
    for all values b in D'_k
      if not consistent(a_{i-1}, x_i = a, x_k = b)
        remove b from D'_k;
    end for
    if D'_k is empty              (x_i = a leads to a dead-end, don't select a)
      reset each D'_k, i < k ≤ n to its value before a was selected;
    else
      return a;
end while
return null
end procedure
```

Constraint Processing,
by R. Dechter
pgs 131-4, 141

44

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**

45

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.

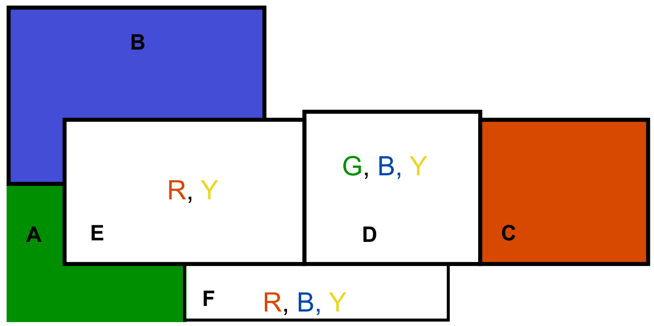
- Least Constraining Value

  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.

46

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).

47

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

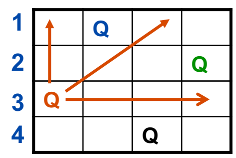
Input: A constraint network R = <X, D, C>
Output: A solution, or notification the network is inconsistent.

$D'_i \leftarrow D_i$ for $1 \le i \le n$;	Copy all domains
$i \leftarrow 1$; $\vec{a}_i = \{\}$	Init variable counter and assignments
$s = \min_{i < j \le n} \lvert D'_j \rvert$	Find unassigned variable w smallest domain
$x_{i+1} \leftarrow x_s$	Rearrange variables so that x_s follows x_i
while $1 \le i \le n$	
instantiate $x_i \leftarrow$ Select-Value-FC();	Select value (dynamic) and add to assignments, $\vec{a}_i$
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
$s = \min_{i < j \le n} \lvert D'_j \rvert$	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

48

Search Performance on N Queens



- **Standard Search**
- **Backtracking**
- **BT with Forward Checking**
- **Dynamic Variable Ordering**
- **Iterative Repair**
- **Conflict-directed Back Jumping**

- A handful of queens
- About 15 queens
- About 30 queens
- About 1,000 queens

49

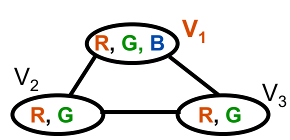
Incremental Repair (Min-Conflict Heuristic)

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 (without or with backup).

R R R: 3	BRR	GRR	RGR	RRG

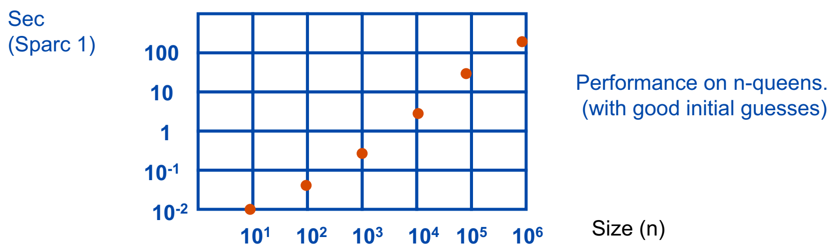


50

Min-Conflict Heuristic

Pure hill climber (w/o backtracking) gets stuck in local minima:

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



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

51

To Solve CSP <X,D,C> We Combine:

1. Reasoning - Arc consistency via 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 That Incorporate Search:

- Standard Search
- Back Track Search (BT)
- BT with Forward Checking (FC)
- Dynamic Variable Ordering (DV)
- Iterative Repair (IR)
- Conflict-directed Back Jumping (CBJ)

52

Next Lecture: 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.

53