Solving Constraint Satisfaction Problems: Search and Forward Checking

Brian C. Williams
16.410-13
October 18th, 2004

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

Reading Assignments: Constraint Satisfaction

Readings:

- Lecture Slides (most material in slides only, READ ALL).
- AIMA Ch. 5 – Constraint Satisfaction Problems (CSPs)
- AIMA Ch. 24.4 pp. 881-884 – Visual Interpretation of line drawings as solving CSPs.

Problem Set #5:

- Covers constraints.
- Online.
- Out Thursday morning, October 14th.
- Extended to Friday, October 22nd.
- Get started early!

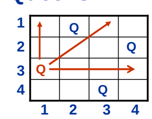
Outline

- Review:
  - Constraint satisfaction problems (CSP)
  - Arc-consistency and propagation
- Analysis of constraint propagation
- Solving CSPs Through Search
- Case Study: Scheduling

CSPS and Encoding 4 Queens

Problem: Place queens so that no two queens can attack each other.

- **Assuming 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 $= \{c_{i,j} \mid i,j \in \{1,2,3,4\}, i \neq j\}$

$c_{i,j}$: Q_i and Q_j on different rows, off diagonal

e.g., $c_{1,2} = \{(1,3)\ (1,4)\ (2,4)\ (3,1)\ (4,1)\ (4,2)\}$

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

Arc Consistency

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

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

$V_i \rightarrow V_j$
$\{2,\cancel{3}\}$ = $\{1,2\}$

$V_i \leftarrow V_j$
$\{2,3\}$ = $\{\cancel{1},2\}$

Achieving Arc Consistency via Constraint Propagation

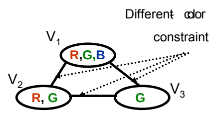
- Directed arc (V_i, V_j) is arc consistent if
  $\forall x \in D_i\ \exists y \in D_j$ such that (x,y) is allowed by constraint C_{ij}

Constraint propagation: To achieve arc consistency of CSP:

1. initialize (fifo) queue with all directed arcs of CSP.
2. For each arc (V_i, V_j) on queue until quiescence:
   a. Delete every value from the tail domain D_i of arc (V_i, V_j) that fails directed arc consistency.
   b. If one or more elements deleted from D_i
      Then add every arc (V_k, V_i) with head V_i to queue (no duplicates).

Constraint Propagation: Graph Coloring

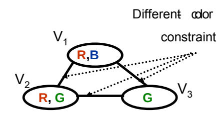
Arc examined	Value deleted
$V_1 > V_2$	
$V_2 > V_1$	
$V_1 > V_3$	
$V_3 > V_1$	
$V_2 > V_3$	
$V_3 > V_2$	



INIT: All arcs on examination queue.
IF Element of variable domain removed,
THEN add all arcs to that variable to queue.

Constraint Propagation: Graph Coloring

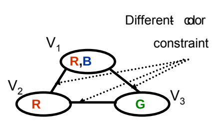
Arc examined	Value deleted
$V_1 > V_2$	none
$V_2 > V_1$	none
$V_1 > V_3$	$V_1 = G$
$V_3 > V_1$	
$V_2 > V_3$	
$V_3 > V_2$	
$V_2 > V_1$	
$V_3 > V_1$	



INIT: All arcs on examination queue.
IF Element of variable domain removed,
THEN add all arcs to that variable to queue.

Constraint Propagation: Graph Coloring

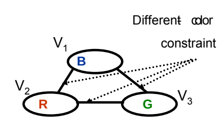
Arc examined	Value deleted
$V_1 > V_2$	none
$V_2 > V_1$	none
$V_1 > V_3$	$V_1 = G$
$V_3 > V_1$	none
$V_2 > V_3$	$V_2 = G$
$V_3 > V_2$	
$V_2 > V_1$	
$V_3 > V_1$	
$V_1 > V_2$	



INIT: All arcs on examination queue.
IF Element of variable domain removed,
THEN add all arcs to that variable to queue.

Constraint Propagation: Graph Coloring

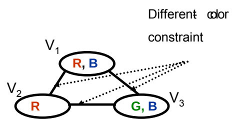
Arc examined	Value deleted
$V_1 > V_2$	none
$V_2 > V_1$	none
$V_1 > V_3$	$V_1 = G$
$V_3 > V_1$	none
$V_2 > V_3$	$V_2 = G$
$V_3 > V_2$	none
$V_2 > V_1$	none
$V_3 > V_1$	none
$V_1 > V_2$	$V_1 = R$
$V_2 > V_1$	none



INIT: All arcs on examination queue.
IF Element of variable domain removed,
THEN add all arcs to that variable to queue.

Solve Graph Coloring 2

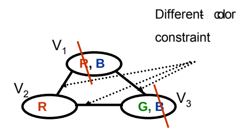
Arc examined	Value deleted
1: $V_1 > V_2$	
2: $V_2 > V_1$	
3: $V_1 > V_3$	
4: $V_3 > V_1$	
5: $V_2 > V_3$	
6: $V_3 > V_2$	



INIT: All arcs on examination queue.
IF Element of variable domain removed,
THEN add all arcs to that variable to queue.

Solution to Graph Coloring 2

Arc examined	Value deleted
1: $V_1 > V_2$	$V_1 = R$
2: $V_2 > V_1$	none
3: $V_1 > V_3$	none
4: $V_3 > V_1$	$V_3 = B$
5: $V_2 > V_3$	none
6: $V_3 > V_2$	none
7: $V_1 > V_3$	none



INIT: All arcs on examination queue.
IF Element of variable domain removed,
THEN add all arcs to that variable to queue.

Outline

- Review:
  - Constraint satisfaction problems (CSP)
  - Arc-consistency and propagation
- Analysis of constraint propagation
- Solving CSPs Through Search

13

What is the Complexity of Constraint Propagation?

Assume:

- Domains are of size at most d.
- There are e binary constraints.

Which is the correct complexity?

1. $O(d^2)$
2. $O(ed^2)$
3. $O(ed^3)$
4. $O(e^d)$

14

Complexity of Constraint Propagation

Assume:

- Domains are of size at most d.
- There are e binary constraints.

Complexity:

- There are 2 * e arcs to check
- Verifying arc consistency takes $O(d^2)$ for each arc.
- An arc is checked at most $O(d)$ times, once for each element of its tail.

⇨ Arc consistency is $O(ed^3)$

15

Is arc consistency sound and complete?

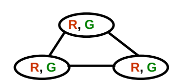
An *arc consistent solution* is any selection of values for every variable from the arc consistent domains.

Completeness: Does arc consistency rule out, as an arc consistent solution, any valid solutions to CSP?

- Yes
- No

Soundness: Is every arc-consistent solution a valid solution to CSP?

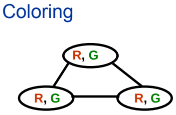
- Yes
- No



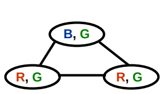
16

Soundness: Arc consistency does not rule out all infeasible candidates

Graph Coloring



arc consistent, but no solutions.



arc consistent, but 2 solutions, not 8.

17

Outline

- Review:
  - Constraint satisfaction problems (CSP)
  - Arc-consistency and propagation
- Analysis of constraint propagation
- Solving CSPs Through Search

18

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 That Incorporate 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

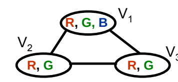
- State
  - Variable assignments thus far
- Initial State
  - No assignments
- Operator
  - New assignment =
    - Select any unassigned variable
    - Select any one of its domain values
  - Child extend assignments with new
- Goal Test
  - All variables are assigned
  - All constraints are satisfied

- Branching factor?

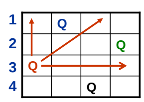
➔ Sum of domain size of all variables O(v*d)

- Performance?

➔ Exponential of branching factor O([v*d]^n)



Search Performance on N Queens



- Standard Search
- Backtracking
- A handful of queens

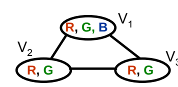
Solving CSPs with Standard Search

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 (i.e., one variable ordering).
2. We can identify a dead end before assigning all variables
   - ➔ Extensions to inconsistent partial assignments are always inconsistent
   - ➔ So check after each assignment.



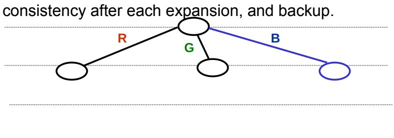
BackTrack Search (BT)

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

V_1 assignments

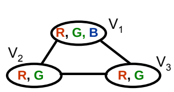
V_2 assignments

V_3 assignments



Preselect order of variables to assign

Expand designated variable



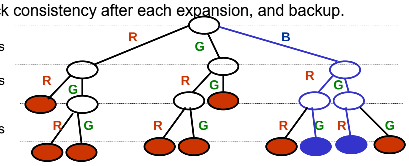
BackTrack Search (BT)

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

V_1 assignments

V_2 assignments

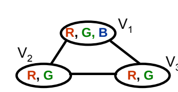
V_3 assignments



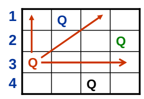
Preselect order of variables to assign

Assign designated variable

Backup at inconsistent assignment



Search Performance on N Queens



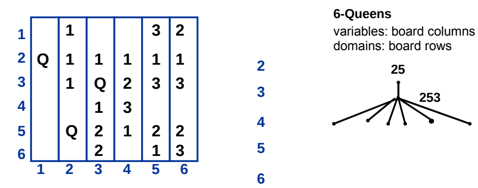
- Standard Search
- Backtracking

- A handful of queens
- About 15 queens

Back jumping

Backtracking At dead end backup to most recent variable,

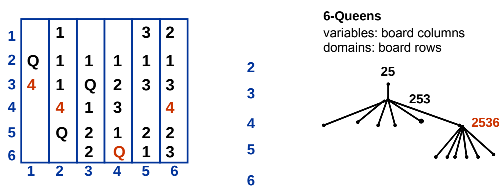
Backjumping At dead end backup to most recent variable that eliminated a value in the current (empty) domain.



Back jumping

Backtracking At dead end backup to most recent variable,

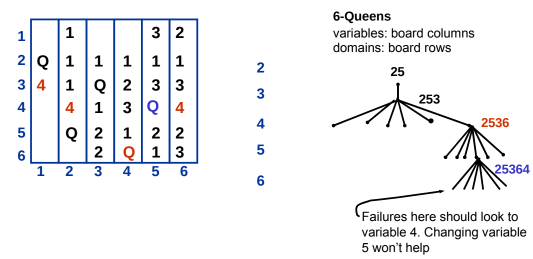
Backjumping At dead end backup to most recent variable that eliminated a value in the current (empty) domain.



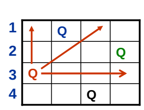
Back jumping

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.



Search Performance on N Queens



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

- A handful of queens
- About 15 queens
- ???

Combine 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 its pruned domain
- Prune (some) domains using constraint propagation
- if a domain has no remaining elements, then backtrack.

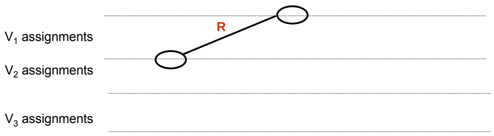
Question: Full propagation is $O(ed^3)$,
How much propagation should we do?

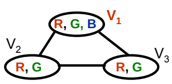
Very little:

- Just check arc consistency for those arcs that terminate 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.

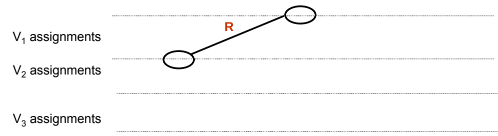


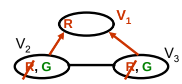


1. Perform initial pruning.

Backtracking with Forward Checking (BT-FC)

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

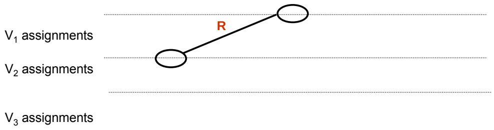


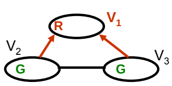


1. Perform initial pruning.

Backtracking with Forward Checking (BT-FC)

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

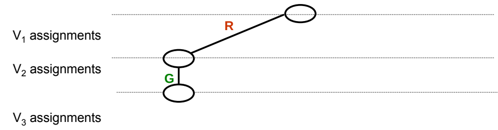


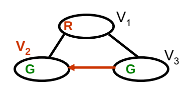


1. Perform initial pruning.

Backtracking with Forward Checking (BT-FC)

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



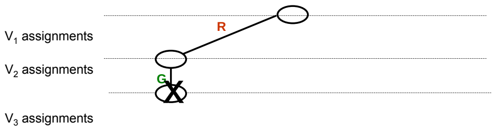


Note: No need to check new assignment against previous assignments

1. Perform initial pruning.

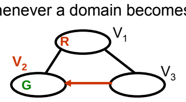
Backtracking with Forward Checking (BT-FC)

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.

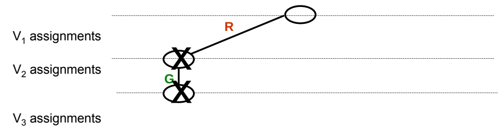
- Back track



1. Perform initial pruning.

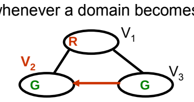
Backtracking with Forward Checking (BT-FC)

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



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

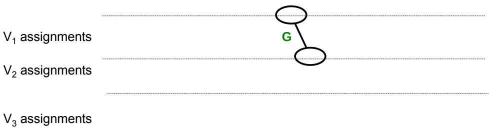
- Back track



1. Perform initial pruning.

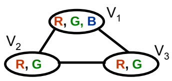
Backtracking with Forward Checking (BT-FC)

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



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

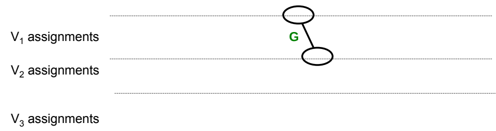
- Back track
- Restore domain values



1. Perform initial pruning.

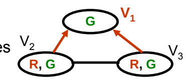
Backtracking with Forward Checking (BT-FC)

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.

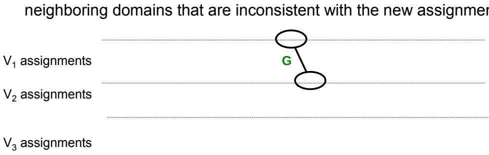
- Back track
- Restore domain values



1. Perform initial pruning.

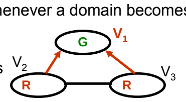
Backtracking with Forward Checking (BT-FC)

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.

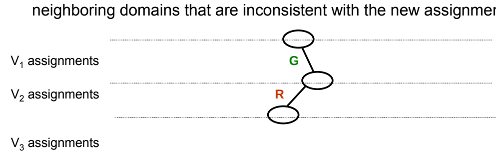
- Back track
- Restore domain values



1. Perform initial pruning.

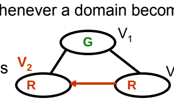
Backtracking with Forward Checking (BT-FC)

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.

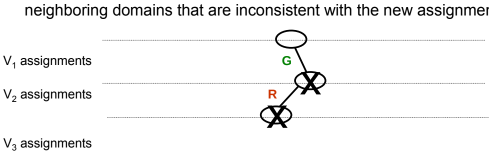
- Back track
- Restore domain values



1. Perform initial pruning.

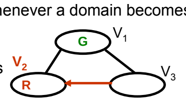
Backtracking with Forward Checking (BT-FC)

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.

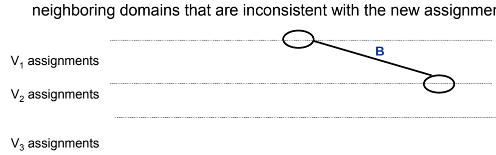
- Back track
- Restore domain values



1. Perform initial pruning.

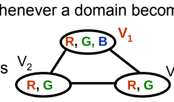
Backtracking with Forward Checking (BT-FC)

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.

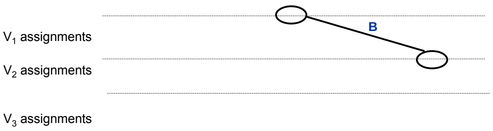
- Back track
- Restore domain values



1. Perform initial pruning.

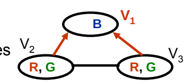
Backtracking with Forward Checking (BT-FC)

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.

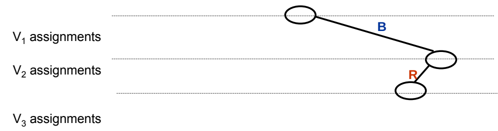
- Back track
- Restore domain values



1. Perform initial pruning.

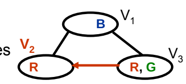
Backtracking with Forward Checking (BT-FC)

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.

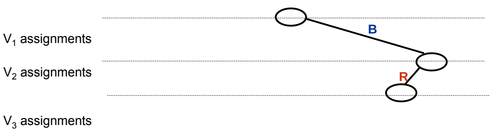
- Back track
- Restore domain values



1. Perform initial pruning.

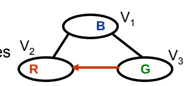
Backtracking with Forward Checking (BT-FC)

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.

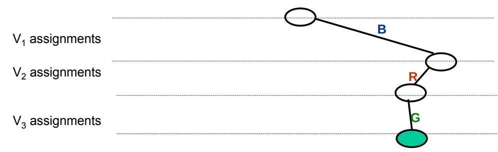
- Back track
- Restore domain values



1. Perform initial pruning.

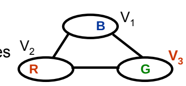
Backtracking with Forward Checking (BT-FC)

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

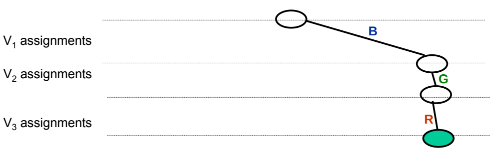


Solution!

1. Perform initial pruning.

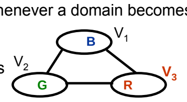
Backtracking with Forward Checking (BT-FC)

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.

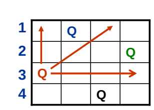
- Back track
- Restore domain values



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**
- **Backjumping**
- **BT with Forward Checking**

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

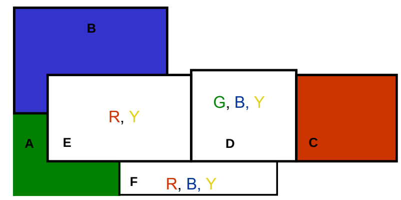
BT-FC with dynamic ordering

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 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 find satisfying assignment.

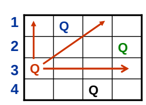
Colors: R, G, B, Y



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



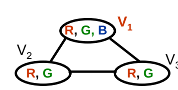
- **Standard Search** • A handful of queens
- **Backtracking** • About 15 queens
- **Backjumping** • ???
- **BT with Forward Checking** • About 30 queens
- **Dynamic Variable Ordering** • About 1,000 queens

Incremental Repair (min-conflict heuristic)

1. Initialize a candidate solution using “greedy” heuristic – get solution “near” correct one.
2. Select a variable in conflict and assign it a value that minimizes the number of conflicts (break ties randomly).

Heuristic used in a local hill-climber (without or with backup).

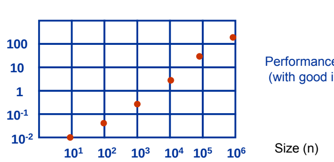
R R R:3	BRR	GRR	RGR	RRG



Min-conflict heuristic

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

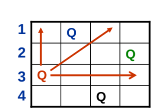
- Add random moves to attempt to get out of minima – generally quite effective.
- Add weights on violated constraints & increase weight every cycle the constraint remains violated.



Performance on n queens.
(with good initial guesses)

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
- **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)

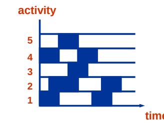
Outline

- Review:
  - Constraint satisfaction problems (CSP)
  - Arc-consistency and propagation
- Analysis of constraint propagation
- Solving CSPs Through Search
- Case Study: Scheduling

Real World Example: Scheduling as a CSP

Choose time for activities:

- Observations on Hubble telescope.
- Jobs performed on machine tools.
- Terms to take required classes.



Variables are activities

Domains sets of possible start times (or "chunks" of time)

Constraints
1. Activities that use the same resource cannot overlap in time, and
2. Preconditions are satisfied.

Case Study: Course Scheduling

Given:

- 40 required courses (8.01, 8.02, ... 6.840), and
- 10 terms (Fall 1, Spring 1, ... , Spring 5).

Find: a legal schedule.

Constraints
- Pre-requisites satisfied,
- Courses offered only on certain terms,
- Limited number of courses taken per term (say 4), and
- Avoid time conflicts.

Note, traditional CSPs are not for expressing (soft) preferences

e.g. minimize difficulty, balance subject areas, etc.

But see recent work on semi-ring CSPs!

Alternative formulations for variables & values

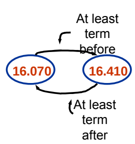
VARIABLES	DOMAINS
A. **1 var per Term** (Fall 1) (Spring 1) (Fall 2) (Spring 2) . . .	All legal combinations of 4 courses, all offered during that term.
B. **1 var per Term-Slot** subdivide each term into 4 course slots: (Fall 1,1) (Fall 1, 2) (Fall1, 3) (Fall 1, 4)	All courses offered during that term.
C. **1 var per Course**	Terms or term-slots. Term-slots make it easier to express the constraint limiting the number of courses per term.

Encoding Constraints

Assume: Variables = Courses, Domains = term-slots

Constraints:

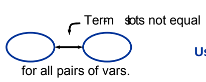
Prerequisite ➨



For pairs of courses that must be ordered.

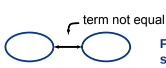
Courses offered only during certain terms ➨ **Filter domain**

Limit # courses ➨



Use term-slots only once

Avoid time conflicts ➨



For course pairs offered at same or overlapping times

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 That Incorporate Search:

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