## AI (in the news)

Click here.

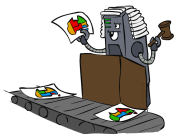
## CS 188: Artificial Intelligence

Constraint Satisfaction Problems II

## Standard Search Formulation

CSP: Variables, domains, and constraints.

Standard search formulation of CSPs



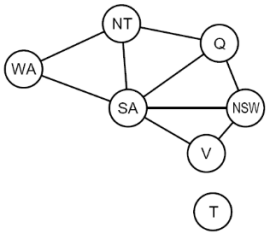
States: partial assignment of values to variables
Initial state: the empty assignment,

Successor function:
Assign unassigned variable

Goal test:
Complete satisfying assignment.

We'll remind ourselves of straightforward, naive approach, and then improve.

## Search Methods



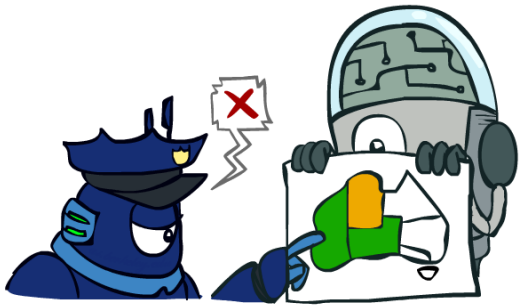

What would BFS do?

What would DFS do?

What problems does naive search have?

## Video of Demo Coloring – DFS

## Backtracking Search



## Backtracking Search

Backtracking search is the basic uninformed algorithm for solving CSPs

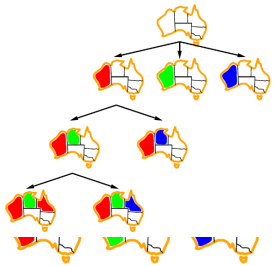
Idea 1: One variable at a time.
- Variable assignments are commutative, so fix ordering
- I.e., [WA = red then NT = green] same as [NT = green then WA = red]
- Assign single variable at each step

Idea 2: Check constraints as you go.
- I.e. consider values which do not conflict with previous assignments
- Might have to do some computation to check the constraints
- "Incremental goal test"

Depth-first search with these two improvements is called **backtracking search** (not the best name)

Can solve n-queens for $n \approx 25$

## Backtracking Example



## Backtracking Search

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
  return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
     if value is consistent with assignment given CONSTRAINTS[csp] then
        add {var = value} to assignment
        result ← RECURSIVE-BACKTRACKING(assignment, csp)
        if result ≠ failure then return result
        remove {var = value} from assignment
  return failure
```

Backtracking = DFS + variable-ordering + fail-on-violation
What are the choice points?

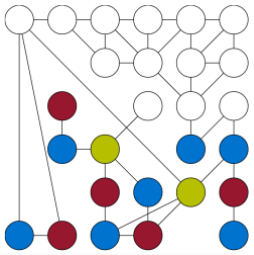
## Video of Demo Coloring – Backtracking

## CSP-Backtracking Search

CSP-Backtracking = DFS + fail-on-violation + variable-ordering

Todo:
- Better "fail on violation". Filtering.
- Pick "better" variable orderings and value orderings.

## An issue.



Consider the partially completed CSP assignment.

Decisions made bottom-up, left-to-right. Let X be the decision is obviously doomed in the current assignment.

What is X?

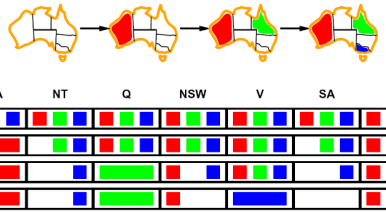
Bonus: How many decisions before CSP-Backtracking search realizes its error?

## Filtering

## Filtering: Forward Checking


Filtering:
Reduce domains for unassigned variables

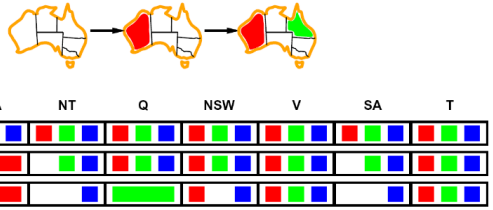
Forward checking:
Remove values that violate constraint in existing assignment




## Video of Demo Coloring – Backtracking with Forward Checking

## Filtering: Constraint Propagation

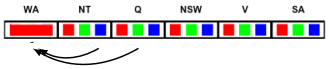
Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:




NT and SA cannot both be blue!
Why didn't we detect this yet?
Constraint propagation:
reason from constraint to constraint

## Consistency of A Single Arc

An arc $X \rightarrow Y$ is consistent iff for every $x$ in the tail there is some $y$ in the head which could be assigned without violating a constraint

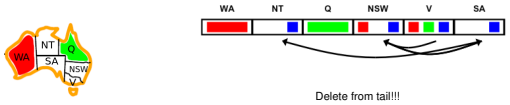




Forward checking: Enforcing consistency of arcs pointing to each new assignment

## Video of Demo Coloring – Backtracking with Forward Checking – Complex Graph

## Arc Consistency of an Entire CSP

A simple form of propagation makes sure all arcs are consistent:




Important: If X loses a value, neighbors of X need to be rechecked!
Arc consistency detects failure earlier than forward checking
Can be run as a preprocessor or after each assignment.

**What's the downside of enforcing arc consistency?**

## Enforcing Arc Consistency in a CSP

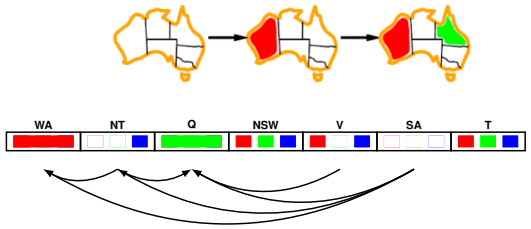
```
function AC-3( csp) returns the CSP, possibly with reduced domains
   inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
   local variables: queue, a queue of arcs, initially all the arcs in csp

   while queue is not empty do
      (Xi, Xj) ← REMOVE-FIRST(queue)
      if REMOVE-INCONSISTENT-VALUES(Xi, Xj) then
         for each Xk in NEIGHBORS[Xi] do
            add (Xk, Xi) to queue
```

```
function REMOVE-INCONSISTENT-VALUES( Xi, Xj) returns true iff succeeds
   removed ← false
   for each x in DOMAIN[Xi] do
      if no value y in DOMAIN[Xj] allows (x,y) to satisfy the constraint Xi ↔ Xj
         then delete x from DOMAIN[Xi];  removed ← true
   return removed
```

Runtime: $O(n^2d^3)$, can be improved to $O(n^2d^2)$.

....but detecting all possible future problems is NP-hard – why?

## Arc Consistency: Step by step.




[Demo: CSP applet (made available by aispace.org) – n-queens]

## Arc Consistency of an Entire CSP

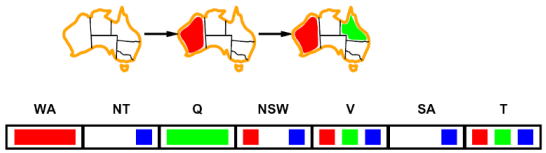
A simple form of propagation makes sure all arcs are simultaneously consistent:

Arc consistency detects failure earlier than forward checking

Important: If X loses a value, neighbors of X need to be rechecked!

Must rerun after each assignment!

Remember: Delete from the tail!



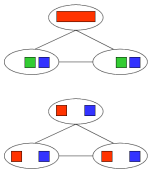

Can also eliminate Blue from NT and SA!

Can backtrack immediately.

## Video of Demo Arc Consistency – CSP Applet – n Queens

## Video of Demo Coloring – Backtracking with Arc Consistency – Complex Graph

## Limitations of Arc Consistency

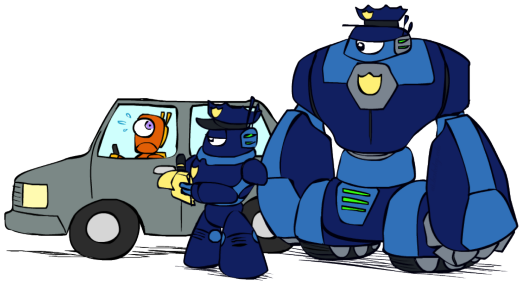


What went wrong here?

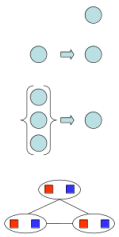
After enforcing arc consistency:
Can have one solution left
Can have multiple solutions left
Can have no solutions left (and not know it)

Arc consistency still runs inside a backtracking search!

## K-Consistency



## K-Consistency



Increasing degrees of consistency

- 1-Consistency (Node Consistency): Each single node's domain has a value which meets that node's unary constraints
- 2-Consistency (Arc Consistency): For each pair of nodes, any consistent assignment to one can be extended to the other
- K-Consistency: For each k nodes, any consistent assignment to k-1 can be extended to the kth node.

Higher k more expensive to compute

(You need to know the k=2 case: arc consistency)

## Strong K-Consistency

Strong k-consistency: also k-1, k-2, . . . 1 consistent

Claim: strong n-consistency means we can solve without backtracking!

Why?

- Choose any assignment to any variable
- Choose a new variable
- By 2-consistency, there is a choice consistent with the first
- Choose a new variable
- By 3-consistency, there is a choice consistent with the first 2
- ...

Lots of middle ground between arc consistency and n-consistency! (e.g. k=3, called path consistency)

## Improving Backtracking

General-purpose ideas give huge gains in speed

Filtering:
Can we detect inevitable failure early?
Forward/Arc/K-consistency.

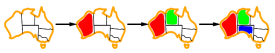
Ordering:
Which variable should be assigned next?
In what order should its values be tried?

Next time. Structure:
Can we exploit the problem structure?

## Ordering

## Ordering: Minimum Remaining Values

Variable Ordering: Minimum remaining values (MRV):
Choose the variable with the fewest legal left values in its domain



Why min rather than max?
Also called “most constrained variable”
“Fail-fast” ordering!

## Ordering: Least Constraining Value

**Value Ordering: Least Constraining Value.**
For a variable, choose the *least constraining value.*
I.e., rules out the fewest values in the remaining variables
Takes computation to determine this! (E.g., rerunning filtering)

Why least rather than most?

All these ordering ideas makes 1000 queens feasible!

## Demo: Coloring – Backtracking + Forward Checking + Ordering

## CSP: what to know.

CSP: variables with domains, constraints.
Model problems: coloring, n-queens, cryptoarithmetic.

Generic Algorithm.
Backtracking.
Filtering. Arc Consistency.
Variable Ordering.Minimum Remaining Values.
Value Ordering. Least Constraining Value.