CONSTRAINT SATISFACTION PROBLEMS

Fabrizio Santini | COMP 131A

VERSION 1.1

TODAY ON AI

- Constraint Satisfaction Problems
- Solving CSPs
- Filtering
- Variable ordering
- Value ordering
- Smart backtracking
- Problem structure
- Questions?

Constraint Satisfaction Problems

SECTION 01

Constraint satisfaction problems (or **CSPs**) belong to a class of problems for which the goal itself is the most important part, not the path used to reach it.

EXAMPLES

- Map coloring!
- Sudokus
- Crossword puzzles
- Job scheduling
- Cryptarithmetic puzzles
- N-Queens problems
- Hardware configuration
- Assignment problems
- Transportation scheduling
- Fault diagnosis
- More…

The state of a CSP is defined by n **variables X_i** with values from **domain D_i**:

- Discrete variables:
  - Domains can be **finite**: a finite of size d set of values or things (means d^n complete assignments). Examples are: Boolean values, specific meaningful numbers, set of colors, etc.
  - or **infinite**: integers or strings. Examples are: strings for a crossword puzzle, duration of jobs in seconds, etc.
- Continuous variables:
  - Domains are **infinite**. Examples are: start/end times for Hubble Telescope observations as they obey to astronomical time laws

- The goal test is a set of **constraints** that specifies allowable combinations of values for subsets of variables:
  - Constraints can be **explicit** (explicitly enumerated)
  - or **implicit** (a formula describes it)
  - Constraints can be: **unary**, **binary**, **global**, **alldiff**

- Constraints are generally represented with a graph, called **hypergraph**, that shows the relationship between the variables

- **Soft constraints** represent preferences about some values of the variables. They usually come with a cost value that expresses the strength of the preference

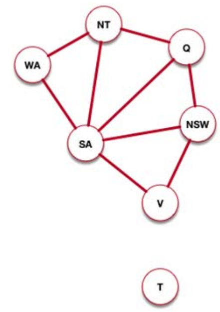
- **VARIABLES**
  WA, NT, Q, NSW, V, SA, T
- **DOMAINS**
  { 🟥 🟩 🟦 }
- **CONSTRAINTS**
  Adjacent regions must have different colors:

  Implicit: WA $\neq$ SA, WA $\neq$ NT, NT $\neq$ SA, NT $\neq$ Q, Q $\neq$ SA, Q $\neq$ NSW, NSW $\neq$ SA, NSW $\neq$ V, V $\neq$ SA

  Explicit: $(WA, NT) \in$ {(🟥,🟩), (🟩,🟦)} etc.

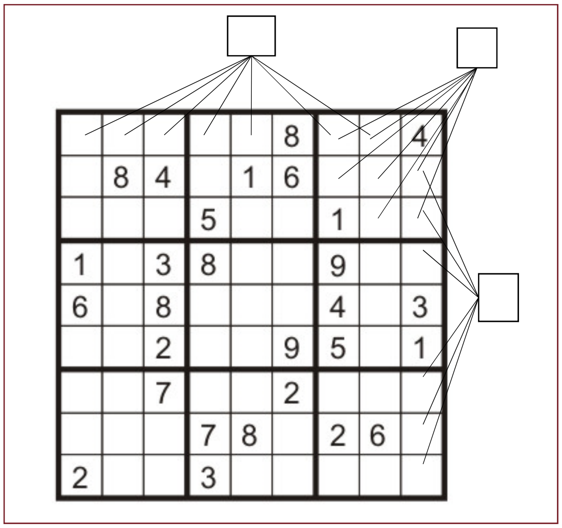
- Solutions are assignments that satisfying all constraints:

WA	🟥	NT	🟩	Q	🟥
NSW	🟩	V	🟥	SA	🟦
T	🟩				



Constraint Satisfaction problems (CSPs)

EXAMPLES: SUDOKU



- **VARIABLES**
  Open squares
- **DOMAINS**
  {1, 2, 3, … 9}
- **CONSTRAINTS**

  9-way alldiff for each column
  9-way alldiff for each row
  9-way alldiff for each region
- **RULES**

  Fill all empty squares so that the numbers 1 to 9 appear once in each row, column and 3x3 box.

Solving CSPs

SECTION 2

The idea is to use standard search algorithms (DFS and BFS) to find a solution that satisfies all the constraints.

- **STATES**
  The variables assigned with values so far
- **INITIAL STATE**
  All variable assignments are empty
- **SUCCESSOR FUNCTION**
  Assign a value to an unassigned variable
- **GOAL TEST**
  The current assignment is complete and satisfies all constraints

It’s a naïve approach, but it’s a start.

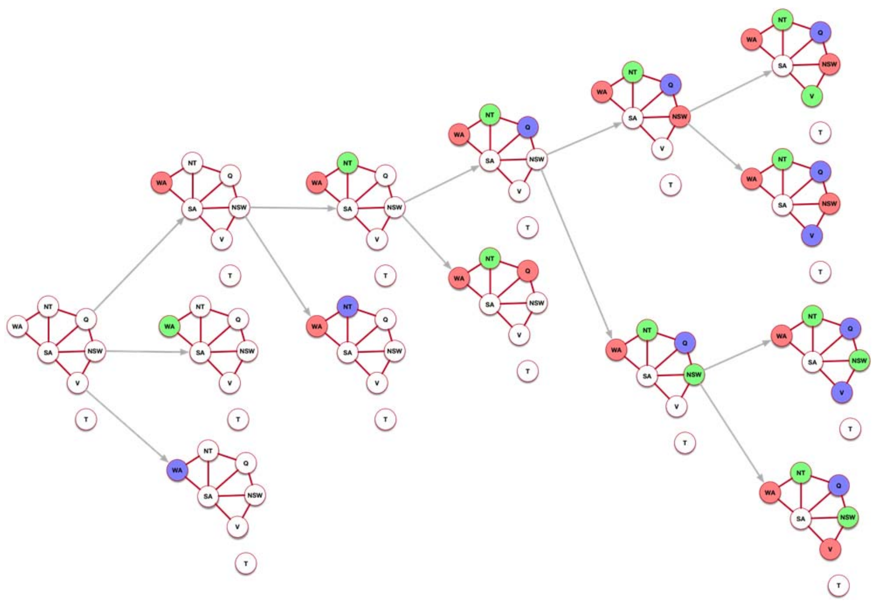
Chronological backtracking search is an uninformed searching algorithm based on the Depth-first searching algorithm with some improvements related to CSPs.

- **IMPROVEMENT 1**
  Each step considers only one assignment at the time

- **IMPROVEMENT 2**
  Check constraints as the search continues. Consider only new assignments which do not conflict previous assignments (incremental goal test)

Backtracking search

COLORING EXAMPLE



Backtracking search

PSEUDO-CODE


```
function Backtracking-search(csp) returns SOLUTION, or FAILURE
  return Recursive-backtracking({}, csp)

function Recursive-backtracking(assignment, csp) returns SOLUTION, or FAILURE
  if assignment is complete then
    return assignment

  variable = Select-unassigned-variable(variables[csp], assignment, csp)
  for each value in Order-Domain-Value(variable, assignment, csp) do
    if value is consistent with assignment given Constraints[csp] then
        add {variable = value} to assignment
        result = Recursive-backtracking(assignment, csp)

        if result ≠ failure then
          return result
        remove {variable = value} from assignment

  return FAILURE
```

We can improve backtracking even more with some additional improvements:

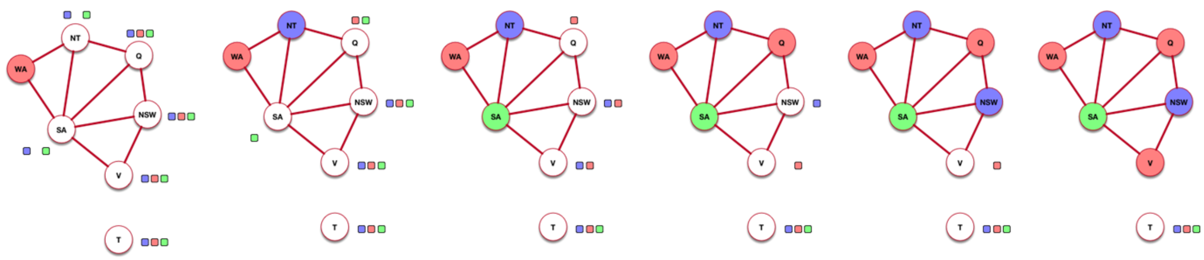
- **IMPROVEMENT 1**
  Taking divination class: filter out inevitable failures as early as possible
- **IMPROVEMENT 2**
  Do not choose poorly: choose carefully which variable for assignment
- **IMPROVEMENT 3**
  You should never, never doubt something that no one is sure of: choose judicially what value to use
- **IMPROVEMENT 4**
  Where we're going we don't need… roads: choose judicially where to backtrack to
- **IMPROVEMENT 5**
  See the whole board: use the topology of the problem, or its structure, to assign variables

Professor Sybill Trelawney: Professor of divination
Indiana Jones: Templar knight
Charlie and the chocolate factory: Roald Dahl
Back to the future: Dr. Emmet Brown
West wing: President Bartlet

Filtering

SECTION 03

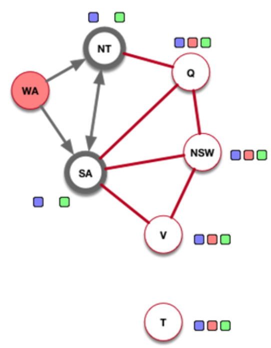
- Forward checking keeps track of the domains for the unassigned variables and remove possible bad options right away

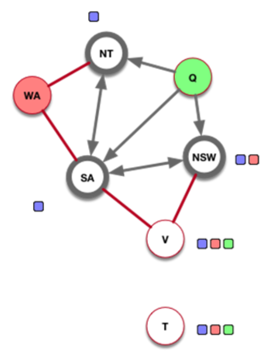


Arc consistency is one form of constraint propagation that tries to prune illegal assignment before they happen

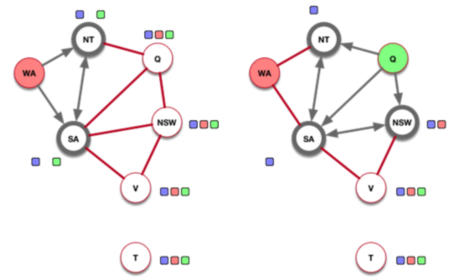
- While evaluating N_Y, an arc $N_X \rightarrow N_Y$ from a neighbor N_X is **consistent** if and only if every $x \in X$ there is some $y \in Y$ which could be assigned without violating a constraint:

Arc consistency on N_X is also triggered if the domain of N_X (X that is) changes.





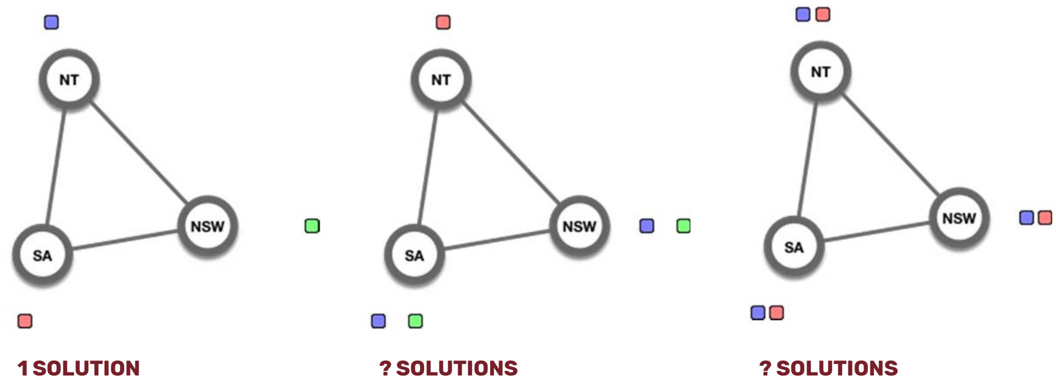
```
function AC-3(csp) returns SOLUTION
push all arcs in queue
while queue is not empty do
  pop arc(X_i,X_j) from queue

  if Remove-Inconsistent-Values(X_i,X_j) then
    for each X_k in Neighbors(X_i) do
      add(X_k,X_i) to queue

function Remove-Inconsistent-Values(X_i,X_j)
  removed = false
  for each x in Domain(X_i) do
    if no value y in Domain(X_j)
        allow (x, y) to satisfy the constraint X_i ↔ X_j then
      delete x from Domain(X_i)
      removed = true
  return removed
```



Filtering

ARC CONSISTENCY CANNOT RUN ALONE

- Arc consistency has to run inside a backtracking search because:
  - There might still be one or more solutions left
  - There might be no solution left



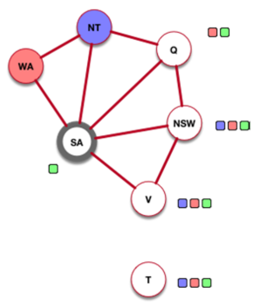
Variable ordering

SECTION 04

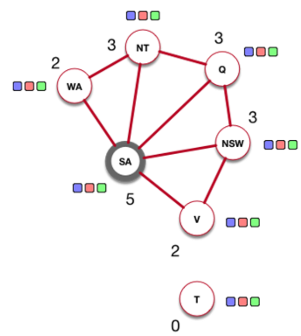
ARTIFICIAL INTELLIGENCE | Ordering
PICKING A VARIABLE

In order to prune even more illegal assignments, we also have to consider how we choose the variables:

Minimum Remaining Values (MRV): choose the variable with the fewest legal values in its domain



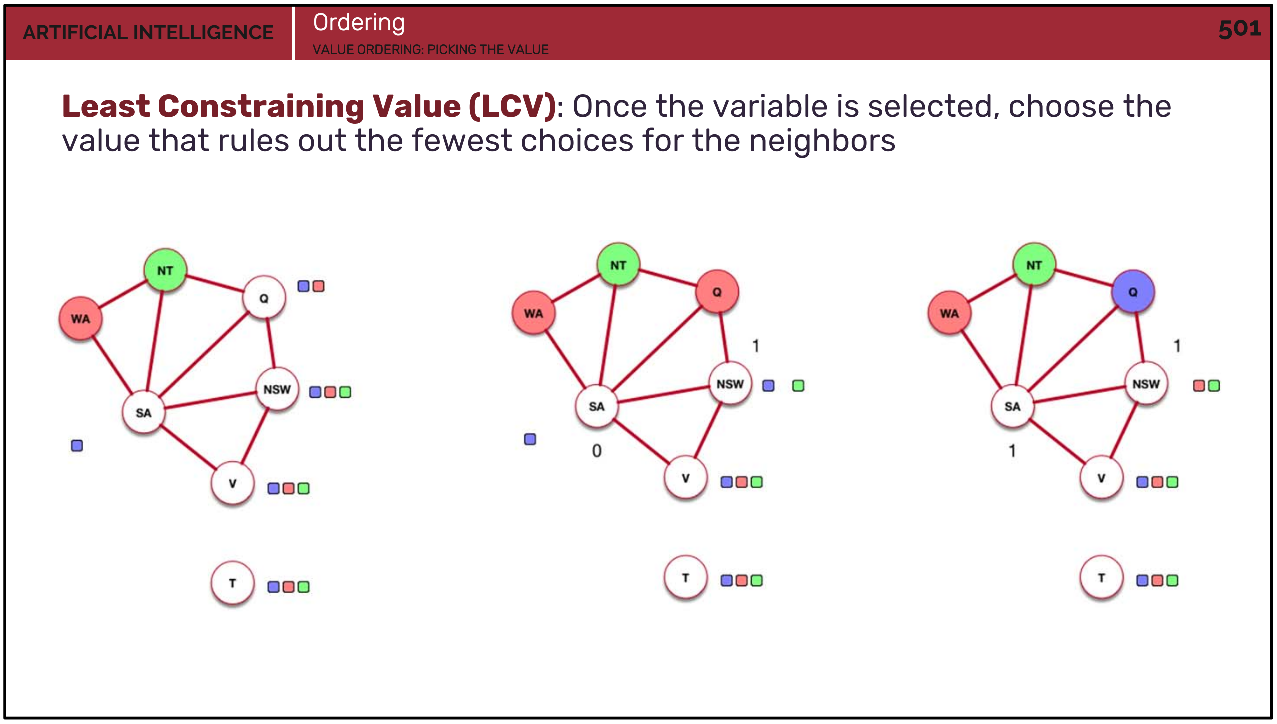
Degree Heuristic: choose the variable with the highest number of constraints



Why do we choose the minimum rather than the maximum?

Value ordering

SECTION 05



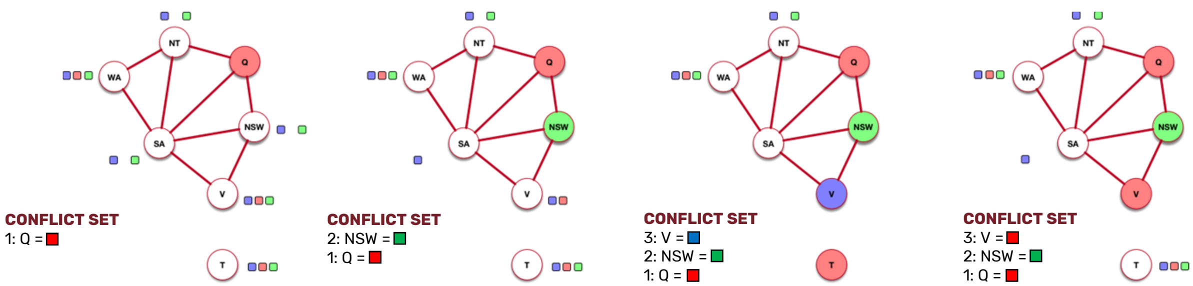
Why do we choose the minimum rather than the maximum?

SECTION 06

Smart backtracking

- The minimal form of backtracking is known as **chronological backtracking**
- **Backjumping**: A smarter approach is to jump back to the most recent conflict using the idea of **conflict sets**

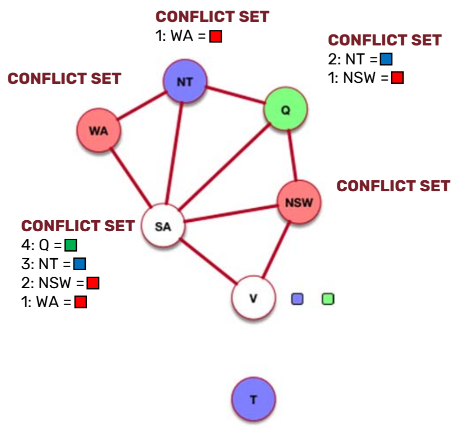
VARIABLE ORDER
Q, NSW, V, T, SA, WA, NT



In **conflict-directed backjumping** the conflict sets migrate from one variable to another:

1. WA =
2. NSW =
3. T =
4. NT =
5. Q =

SA = ?



SA

CONFLICT SET
4: Q =
3: NT =
2: NSW =
1: WA =

Q

CONFLICT SET
2: NT =
1: NSW =

CONFLICT SET
3: NT =
2: NSW =
1: WA =

NT

CONFLICT SET
1: WA =

CONFLICT SET
2: NSW =
1: WA =

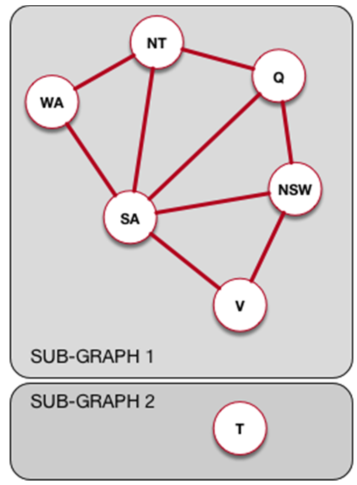
NSW

Problem structure

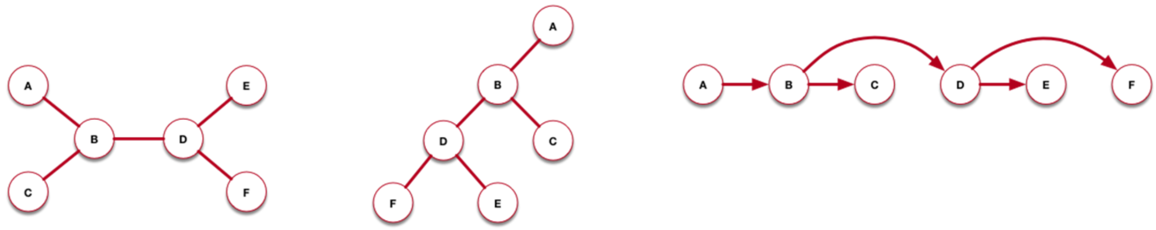
SECTION 07

Independent sub-problems can make life much easier:

- The worst-case complexity of a solution search is normally $O(d^n)$. For a problem with $n = 60$ and $d = 2$ (a binary domain), and assuming 1M node/s evaluation, the search takes 36,558 years
- In the case the problem can be broken into smaller problems with c variables, worst-case complexity is $O\left(\frac{n}{c}d^c\right)$. For the same problem above, with $c = 20$, the search would only be 3s
- Independent sub-problems are identifiable as connected components of the constraint graph



If the constraint graph has no loops, the CSP can be solved in $O(nd^2)$.



1. **Remove backward**: Apply arc consistency from the deepest leaf to its parent. After this phase, all arcs are consistent.
2. **Assign forward**: Assign a value to the variable consistent with its parent. Forward assignment will never backtrack.

Sometimes is possible to find one or more variables that, if instantiated, transform the constraint graph into a tree. This process is called **cutset conditioning**.

With a cutset of size c, complexity of nearly tree-structured CSPs is $O(d^c(n-c)d^2)$

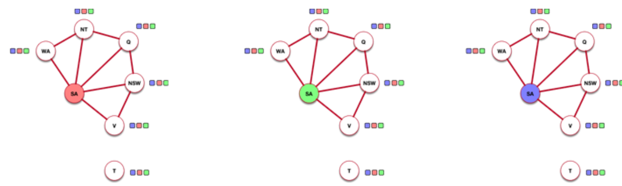
The process requires to instantiate the variables of the cutset and prune its neighbors' domain.

ARTIFICIAL INTELLIGENCE

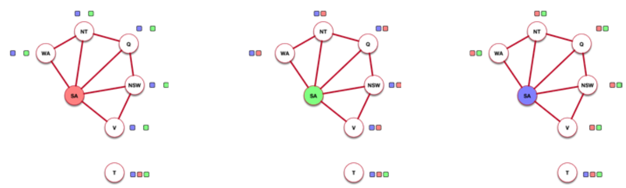
Problem structure

NEARLY TREE-STRUCTURED CSPS: EXAMPLE

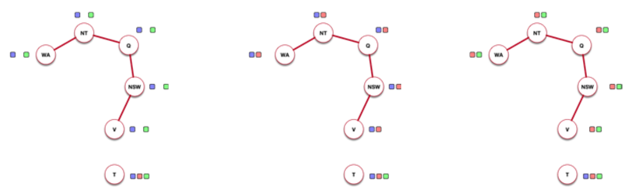
INSTANTIATE THE CUTSET



PRUNE THE REMAINING DOMAINS



SOLVE THE RESIDUAL CSPS



QUESTIONS ?