

## Constraint Satisfaction Problem

Identification problem: These are problems in which we must simply identify whether a state is a goal state or not


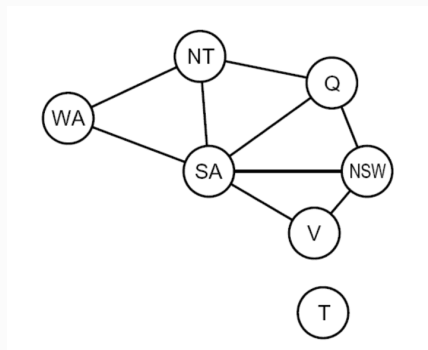
- State is defined by
  - Variables  $X_i$
  - with values from Domain  $D$
- Goal test is a set of constraints

CSPs are represented as constraint graphs, where nodes represent variables and edges represent constraints between them.

### Constraints

Type	Example
Implicit	$A \neq B$
Explicit	$(A, B) \in \{ (red, green), (red, blue), ... \}$

### Graph Coloring

Problem	Constraint Graph
	

##

Variables	WA, NT, Q, NSW, SA, V, T
Domain	{Red, Green, Blue}
Constraints	$WA \neq NT$ ...

# N-Queens

	Formulation 1	Formulation 2
Variables	$X_{ij}$	$Q_k$
Domain	$\{0, 1\}$	$\{1, 2, \dots, N\}$
Constraints	$\sum_{i,j} X_{ij} = N$ $\forall i, j, k \quad (X_{ij}, X_{ik}) \in \{(0, 0), (0, 1), (1, 0)\}$ $\forall i, j, k \quad (X_{ij}, X_{kj}) \in \{(0, 0), (0, 1), (1, 0)\}$ $\forall i, j, k \quad (X_{ij}, X_{i+k, j+k}) \in \{(0, 0), (0, 1), (1, 0)\}$ $\forall i, j, k \quad (X_{ij}, X_{i+k, j-k}) \in \{(0, 0), (0, 1), (1, 0)\}$	Implicit: $\forall i, j$ non threatening $(Q_i, Q_j)$ Explicit: $(Q_{i1}, Q_{i2}) \in \{(1, 3), (1, 4), \dots\}$

## IDK

- Binary CSP: Each constraint relates at most 2 variables
- Binary constraint graph: nodes are variables, arcs show constraints

## Cryptarithmic

Variables	
Domain	$[0, 9]$
Constraints	alldiff(variables)

## Sudoku

Variables	$X_{ij}$
Domain	$[1, 9]$
Constraints	9-way all diff for each column 9-way all diff for each row 9-way all diff for each sub-grid

## Types

Variable Type	Domain		Examples
Discrete	Finite	Size $d$ means $O(d^n)$ complete assignments	Boolean satisfiability (np-complete)
	Infinite (integers, strings)		Job Scheduling (Vars are start/end times for each job) Linear constraints solvable Non-linear undecidable
Continuous		Linear constraints solvable in polynomial time by LP methods	Start/end times for Hubble telescope observations

## Constraints

Variety		Example
Unary	Single variable	SA $\neq$ Green
Binary	Pairs	SA $\neq$ WA
Higher-Order		Cryptarithmic column constraints

Enforcement		Example
Soft (Preferences)	Represented by cost for each var assignment Gives constrained optimization problems	Red better than green
Hard		

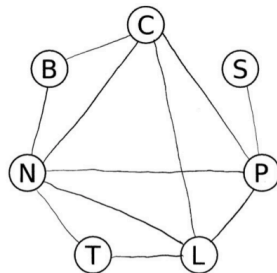
## Standard Search Formulation

States are defined by the values assigned so far (partial assignments)

- Initial state: Empty assignment
- Successor function: assign value to unassigned variable
- Goal test: current assignment is complete and satisfies all constraints

## IDK

Constraint graph for this problem



Domains for this problem

T	1
L	1 2 3 4
B	1 2 3 4
C	1 2 3 4
S	1 2 3 4
P	1 2 3 4
N	1 2 3 4

## Backtracking Search

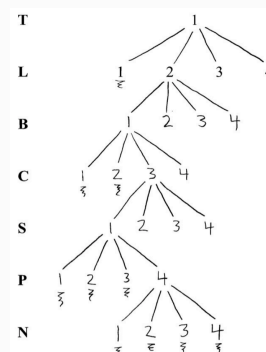
Backtracking = DFS + variable-ordering + fail-on-violation

Assumption: assignments are commutative (order of assignment doesn't matter)

1. Fix an ordering for variables, and select values for variables in this order
2. Consider assignments to a single var at each step
3. Check constraints on the go
  - When selecting values for a variable, only select values that don't conflict with any previously assigned values
  - If no such values exist, backtrack and return to the previous variable, changing its value

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

ex	Var assigned or de-queued	List all values eliminated from neighboring variables	Back track ?	ex	Var assigned or de-queued	List all values eliminated from neighboring variables	Back track ?
	X	Y ≠ 3, 4   Z ≠ 3 (example)	<input checked="" type="checkbox"/>	10	P = 2	NONE	<input checked="" type="checkbox"/>
1	T = 1	NONE	<input type="checkbox"/>	11	P = 3	NONE	<input checked="" type="checkbox"/>
2	L = 1	NONE	<input checked="" type="checkbox"/>	12	P = 4	NONE	<input type="checkbox"/>
3	L = 2	NONE	<input type="checkbox"/>	13	N = 1	NONE	<input checked="" type="checkbox"/>
4	B = 1	NONE	<input type="checkbox"/>	14	N = 2	NONE	<input checked="" type="checkbox"/>
5	C = 1	NONE	<input checked="" type="checkbox"/>	15	N = 3	NONE	<input checked="" type="checkbox"/>
6	C = 2	NONE	<input checked="" type="checkbox"/>	16	N = 4	NONE	<input checked="" type="checkbox"/>
7	C = 3	NONE	<input type="checkbox"/>	17			<input type="checkbox"/>
8	S = 1	NONE	<input type="checkbox"/>	18			<input type="checkbox"/>
9	P = 1	NONE	<input checked="" type="checkbox"/>	19			<input type="checkbox"/>



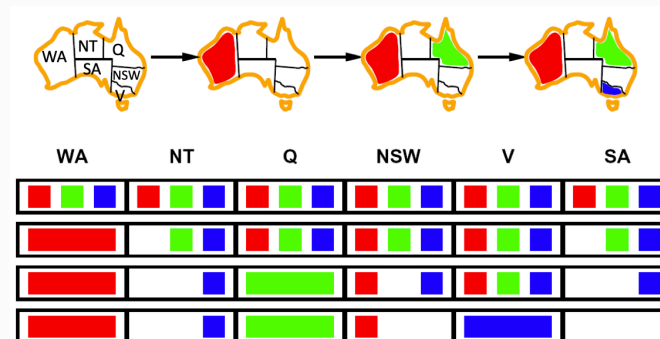
## Filtering/Pruning

To improve performance, we consider filtering which checks if we can prune the domain of unassigned variables ahead of time.

To improve performance, we can prune subtrees that will inevitably lead to failure

## Forward Checking

- Whenever a new variable is assigned, we can run forward checking and prune the domains of unassigned variables adjacent to the newly assigned variable in the constraint graph.
- Basically we eliminate all the values from the domain of the adjacent variables which could cause violation of any constraint.



This propagates info from assigned to unassigned vars, but doesn't provide early detection for all failures

Time Complexity:  $O(n^2d^3)$

	Var assigned or de- queued	List all values eliminated from neighboring variables	Back track ?		Var assigned or de-queued	List all values eliminated from neighboring variables	Back track ?
ex	X	Y ≠ 3, 4   Z ≠ 3   (example)	<input checked="" type="checkbox"/>	7	S = 2	NONE	<input type="checkbox"/>
1	T = 1	L ≠ 1   N ≠ 1	<input type="checkbox"/>	8	P = 1	NONE	<input type="checkbox"/>
2	L = 2	P ≠ 2   N ≠ 2   C ≠ 2	<input type="checkbox"/>	9	N = 4	NONE	<input type="checkbox"/>
3	B = 1	C ≠ 1	<input type="checkbox"/>	10			<input type="checkbox"/>
4	C = 3	N ≠ 3   P ≠ 3	<input type="checkbox"/>	11			<input type="checkbox"/>
5	N	P ≠ 4	<input type="checkbox"/>	12			<input type="checkbox"/>
6	P	S ≠ 1	<input type="checkbox"/>	13			<input type="checkbox"/>

## Arc Consistency

An arc  $X \rightarrow Y$  is consistent  $\iff \forall x$  in the tail,  $\exists y$  in the head which could be assigned without violating a constraint

- Forward checking only enforces consistency of arcs pointing to each new assignment
- More advanced: 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 pre/post-processing step for each assignment

Note: delete from tail

Time Complexity:  $O(n^2 d^2)$

But detecting all possible future problems is np-hard

### Limitations

- After enforcing arc consistency
  - Can have one solution left
  - Can have multiple solutions left
  - Can have no solutions left (and not know about it)
- Arc consistency still runs inside a backtracking search

### AC3 Algorithm



1. Turn each binary constraint represented as undirected edge into 2 directed arcs

Eg

- $A \neq B \implies A \neq B, B \neq A$
- $A < B \implies A < B, B > A$

2. Add all arcs to agenda  $Q$
3. Repeat until  $Q$  empty
  1. Take an arc  $(X_i, X_j)$  off  $Q$  and check it
  2.  $\forall X_i, \exists X_j$ : For every element of  $X_i$  there should be at least one element of  $X_j$  that satisfies condition
  3. Remove any inconsistent values from  $X_i$
  4. if  $X_i$  has changed, add all arcs of the form  $(X_k, X_i)$  to agenda
    1. If arc  $X_k \rightarrow X_i$  is already in  $Q$ , don't add it again

*Ordering*

Ordering		Disadvantage
MRV: Minimum Remaining Values/ “Fail-Fast”	Choose “most constrained var”, ie the var with the fewest legal left values in domain	
LCV: Least Constraining Value	Choose least constraining value Ie, var that rules out the fewest values in the remaining vars	Extra computation for re-running filtering
Degree	Choose node with highest degree  Choose var involved in most no of constraints on other unassigned vars	
Min- Conflicts	chooses randomly any conflicting variable, i.e., the variable that is involved in any unsatisfied constraint, and then picks a value which minimizes the number of violated constraints (break ties randomly)	