Constraint Satisfaction Problems

In which we see how treating states as more than just little black boxes leads to the invention of a range of powerful new search methods and a deeper understanding of problem structure and complexity.

Content

1. Constraint Satisfaction Problems (CSP)
2. Backtracking-Search for CSPs
3. Local Search for CSPs
4. The structure of problems
5. Summary

Constraint-Problems

- Regular search problems
  - Structure of states of problem space not used by search algorithms.
  - Goal state is either valid or invalid.

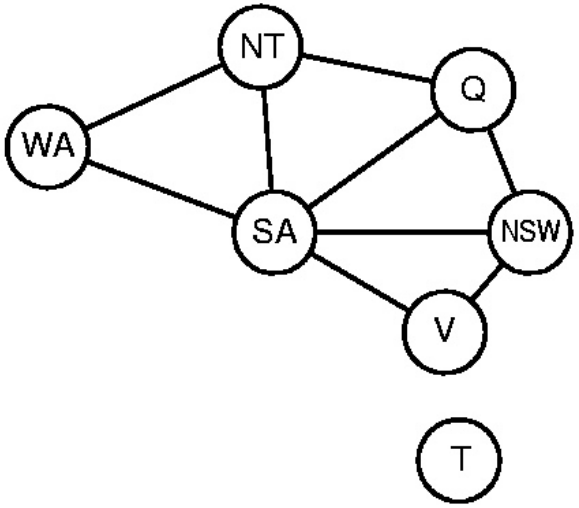
- Constraint Satisfaction Problem
  - Goal condition consists of a set of sub-conditions. All have to be valid in a goal state.
  - Difference: satisfiable / partly satisfiable / not satisfiable.
  - Constraint-Solver make use of this difference.

Constraint Satisfaction Problem

- Defined by
  - a set of variables $X_1, X_2, \ldots, X_n$
  - a set of constraints $C_1, C_2, \ldots, C_m$
- Each variable X_i has a non-empty domain D_i of possible values.
- Each constraint C_i involves some subset of the variables and specifies the allowable combinations of values for that subset.

- A state of the problem is defined by an assignment of values to some or all of the variables $\{X_i=v_i , X_j=v_j , \ldots\}$.
- An assignment that does not violate any constraints is called a consistent or legal assignment.
- A complete assignment is one in which every variable is mentioned, and a solution to a CSP is a complete assignment that satisfies all the constraints.
- For some CSPs any solution will do, but others require a solution that maximizes an objective function.

Example: map coloring

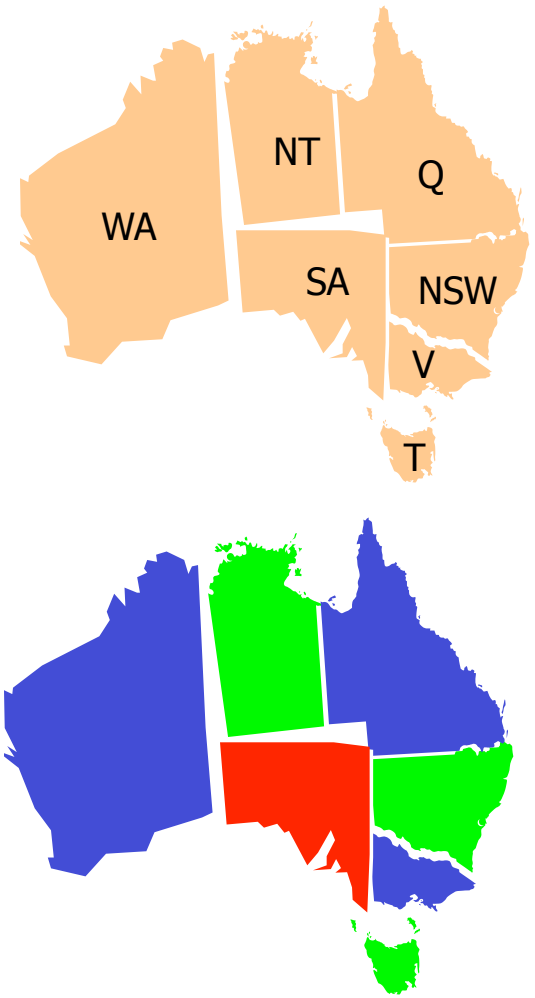


- Coloring of states can be defined as CSP.
- Goal: coloring each region either red, green, or blue in such a way that no neighboring regions have the same color.

- Constraint-Graph (CG)
  - variables as nodes
  - constraints as arcs

Example: map coloring

- Variables are regions
  - WA, NT, Q, NSW, V, SA, T
- Domain for each variable
  - {red, green, blue}
- Constraints
  - E.g. for WA and NT
    {(red,green),(red,blue),(green,red),
    (green,blue), (blue,red),{blue,green)}
- Multiple solutions, e.g.
  - {WA=blue, NT=green, Q=blue,
    NSW=green, SA=red, T=green, V=blue}



Properties

- Treat problems as CSPs has advantages.
- Representation of states as patterns (set of variables with values).
  - Thus, successor function can be described generic.
  - Effective heuristics can be developed, because additional domain knowledge is not needed.
  - Finally, the structure of the constraint graph can be used to simplify the solution process, in some cases giving an exponential reduction in complexity.
- CSP can be given an incremental formulation as a standard search problem as follows:
  - Initial state: the empty assignment {}, i.e., all variables are unassigned.
  - Successor function: a value can be assigned to any unassigned variable, provided it does not conflict with previously assigned variables.
  - Goal test: the current assignment is complete.
  - Path cost: a constant cost (e.g., 1) for every step.
- Every solution must be a complete assignment and therefore appears at depth *n* if there are *n* variables.
- Depth- first search algorithms are popular for CSPs

Properties (2)

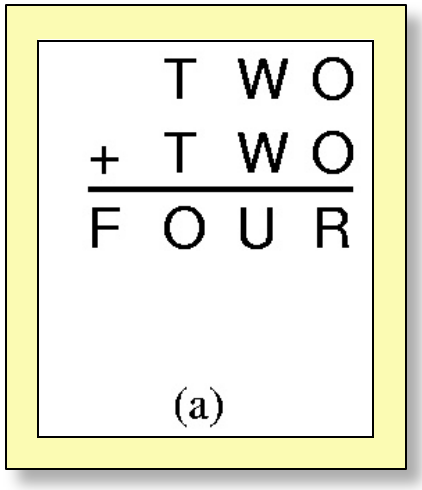
- Path by which a solution is reached is irrelevant.
- Simplest kind of CSP involves variables that are **discrete** and have **finite domains**.
- E.g.: map-coloring, 8-queens problem
- If max(domain) = d, then the number of possible complete assignments is $O(d^n)$ — that is, exponential in the number of variables.
- Finite domains include Boolean CSPs
- Boolean CSPs can be NP-complete (e.g., 3SAT problem)
- In the worst case, therefore, we cannot expect to solve finite-domain CSPs in less than exponential time.
- In most practical applications, however, general-purpose CSP algorithms can solve problems orders of magnitude larger than those solvable using the general-purpose search algorithms.

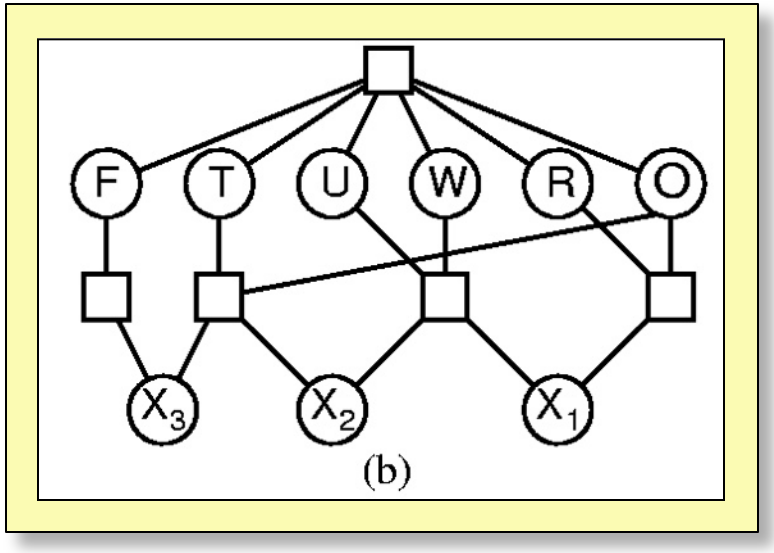
Properties (3)

- Discrete variables can also have infinite domains—for example $\mathbb{N}$
- Example: Job scheduling
  - With infinite domains, no sense to list all combinations→ use constraint language
  - Job_1 needs 5 days and has to precede Job_3, use algebraic description: StartJob1 + 5 < StartJob3
- Special solution algorithms exist for linear constraints on integer variables—that is in which each variable appears only in linear form,
- None for continuous constraints
- Constraint satisfaction problems with continuous domains are very common in the real world and are widely studied in the field of OR.
- Example: Timing of experiments for the Hubble Telescope

Types of Constraints

- Unary, assigns one value to one variable (e.g. SA = red)
- Binary, (e.g. SA-NSW), can be represented as CG
- High-order constraints have three or more variables (e.g. crypt-arithmetic-problem)
  - Each letter represents a digit
  - alldiff(F,T,U,W,R,O)
  - Constraints:
    $O + O = R + 10 * X_1$
    $X_1 + W + W = U + 10 * X_2$
    $X_2 + T + T = O + 10 * X_3$
    $X_3 = F$
    with
    X_1, X_2, X_3 as auxiliary variables {0,1}
  - Constraint Hypergraph
- Each high-order constraint can be transformed into multiple binary constraints
- Preferences via costs

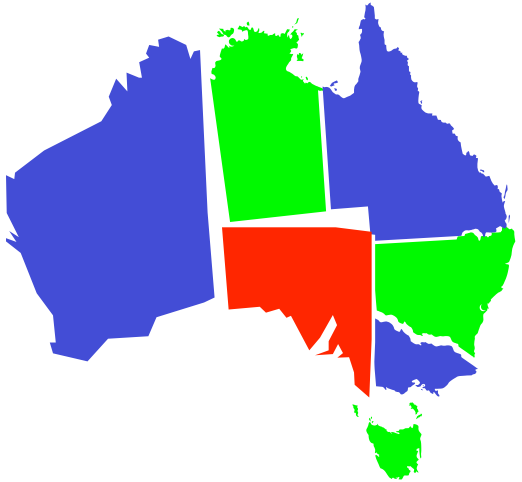




Commutativity

- Each search algorithm from chapters 3+4 can be used.
- Assumption: breath-first, *b* on level 0 is *nd*, because each of the *d* values can be assigned to each of the variables.
- Next level: *(n-1)d*, i.e. $n!d^{n,}$ despite the d^n possible assignments.
- → Important feature: **Commutativity**
- A problem is commutative if the order of application of any given set of actions has no effect on the outcome.

- → Successor: all possible assignments only for ONE variable per node
- Example: SA=red, SA=blue, but not SA=red, NSW=green
- With this restriction d^n



Backtracking-Search for CSPs

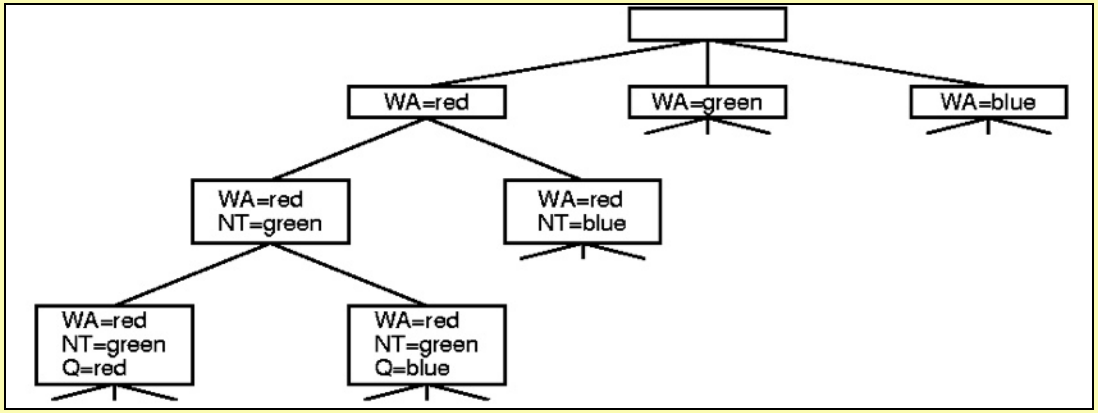
- The term backtracking search is used for a depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign.

- Differences:
  - Expansion of assignment, no copying
  - No initial state, successor function, goal test

- Part of search tree

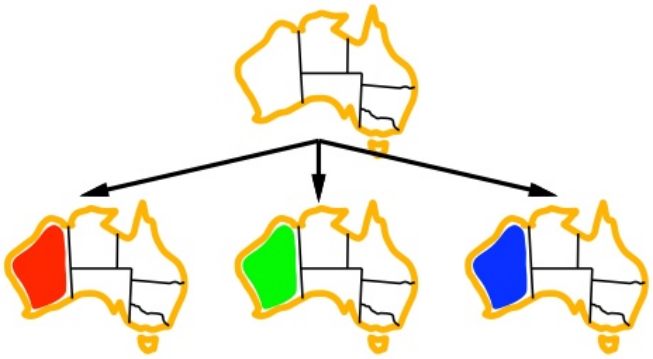
```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns a solution, or 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 according to CONSTRAINTS[csp] then
      add {var = value} to assignment
      result ← RECURSIVE-BACKTRACKING(assignment, csp)
      if result ≠ failure then return result
      remove {var = value} from assignment
  end
  return failure
```

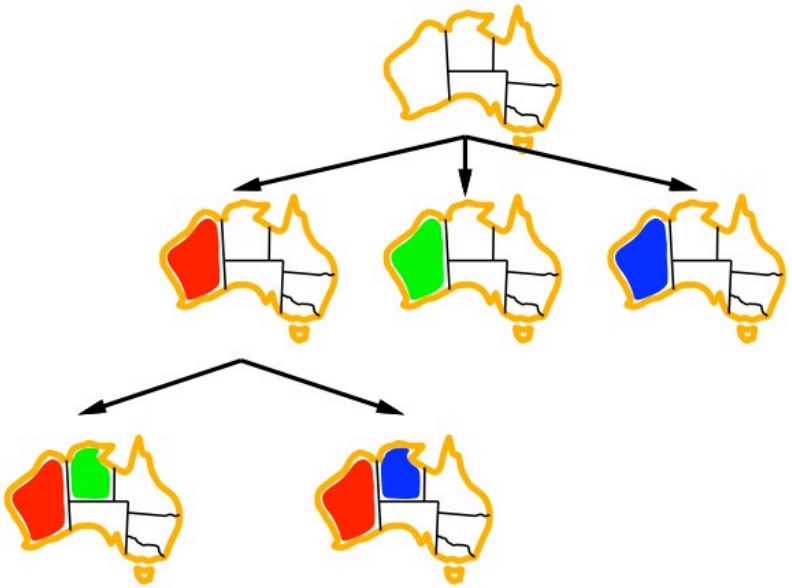


Backtracking example

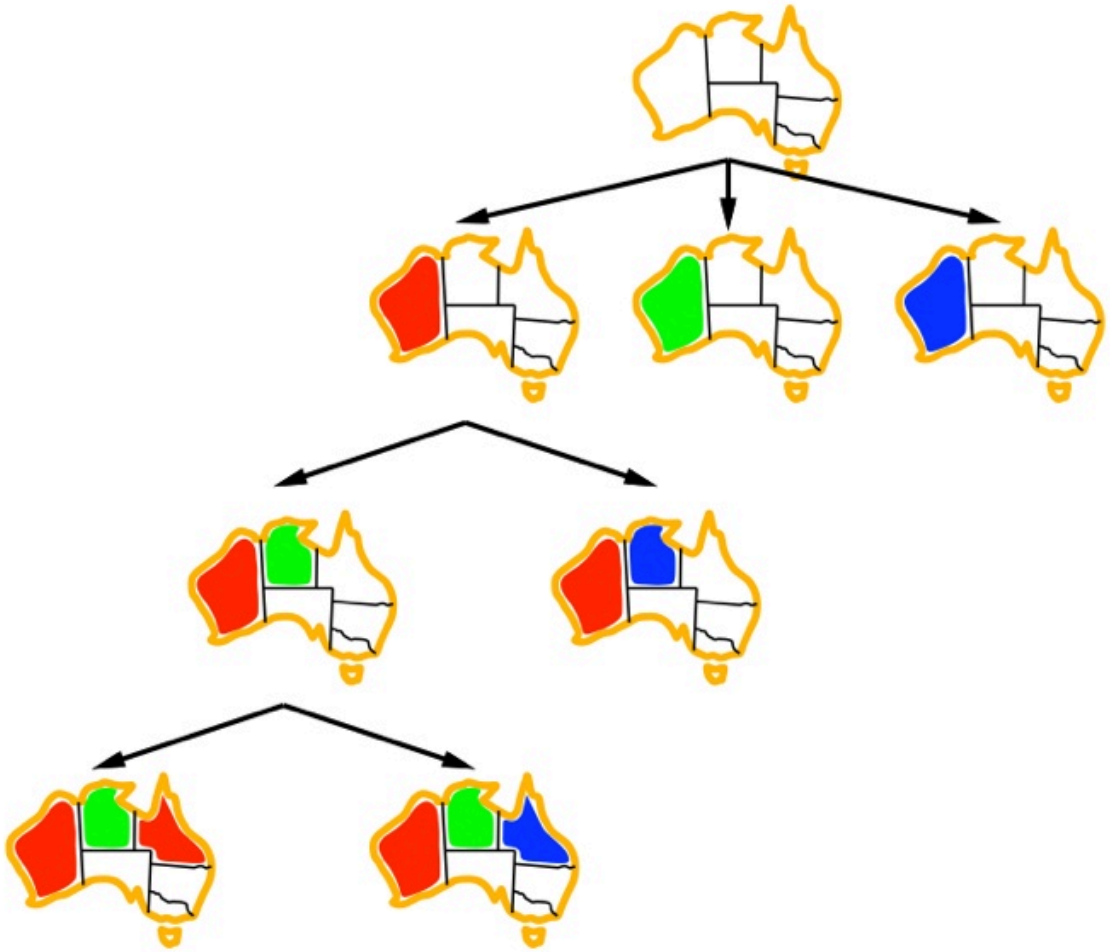
Backtracking example



Backtracking example



Backtracking example



Ranking of variables and values

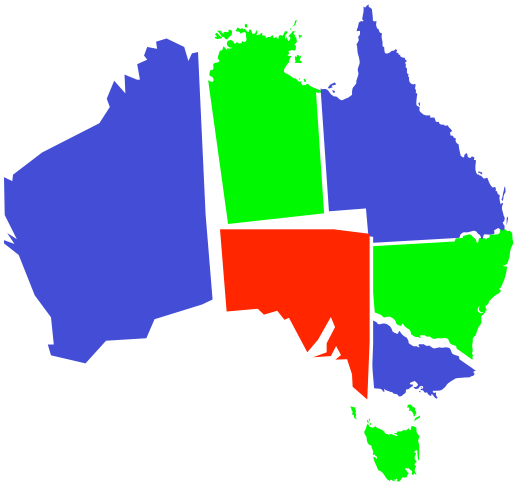
- Plain backtracking is an uninformed algorithm, so we do not expect it to be very effective for large problems.
- Knowledge about problems (*h*) can help to make search more efficient.
- Here: efficiency without domain-specific knowledge
- Instead methods, that can answer the following questions:

1. Which variable should be assigned next and in what order should its values be tried?
2. What are the implications of the current variable assignments for the other unassigned variables?
3. When a path fails—that is, a state is reached in which a variable has no legal values— can the search avoid repeating this failure in subsequent paths?

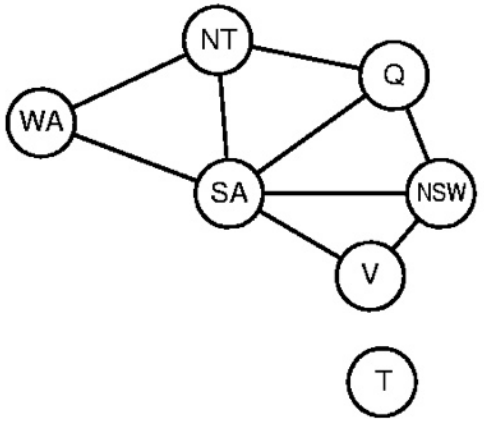
Ranking of variables and values

$var \leftarrow$ SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], $assignment$, csp)

- Minimum Remaining Values (MRV)
  - Variable with least legal values first
  - Ex: WA=blue, NT=green → only one option for SA (red)
  - SA instead of Q
  - Factor 3-3000 better than BT
  - also: heuristic of the least restricting variable

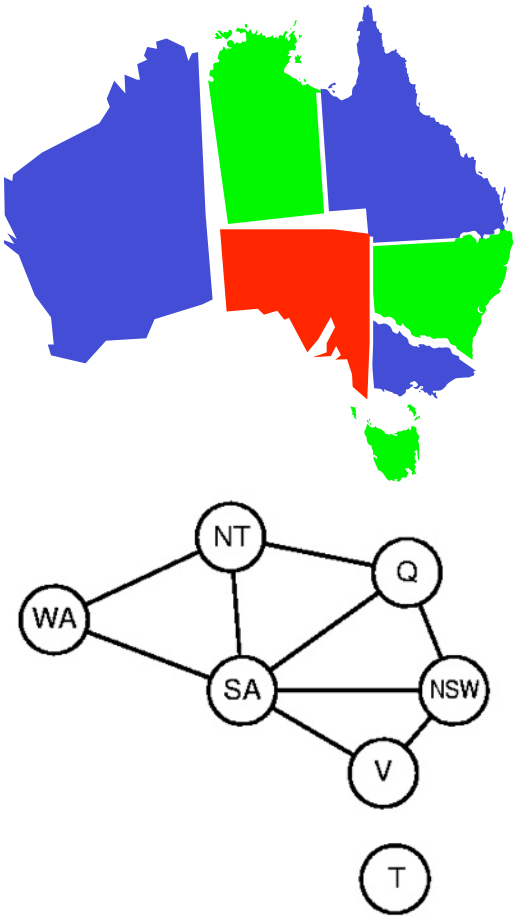


- MRV does not do well in the beginning → Degree Heuristic
  - Reduction of *b* through selection of variable that is involved in most constraints to other free variables
  - Ex.: SA=5, T=0, NSW=3



Ranking of variables and values

- Least Constraining Value (LCV)
  - Variable is selected
  - Values have to be assigned
  - Preference: Value that leaves the least choices for neighbor variable
  - Ex.: suppose WA=blue, NT=green, next choice is Q: Red no good → last legal choice for SA, thus blue
- In general: LCV Heuristic tries to keep max flexibility for assignments

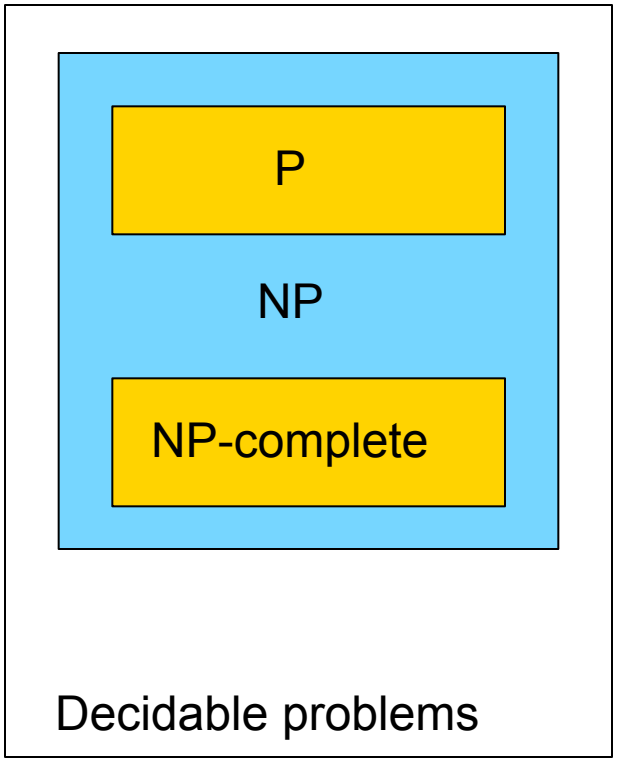


Complexity

- Complexity of algorithms
  - Worst-case behavior, e.g. sorting in $O(n^2)$
  - Delivers upper bound for best algorithm
  - Does not show that there might be a better algorithm, e.g. sorting in O(n log n)
- Complexity of problems
  - Worst-case behavior
  - Coarse classification
  - Efficiently solvable: tractable
  - Not efficiently solvable: intractable
  - Delivers lower bound for best algorithm

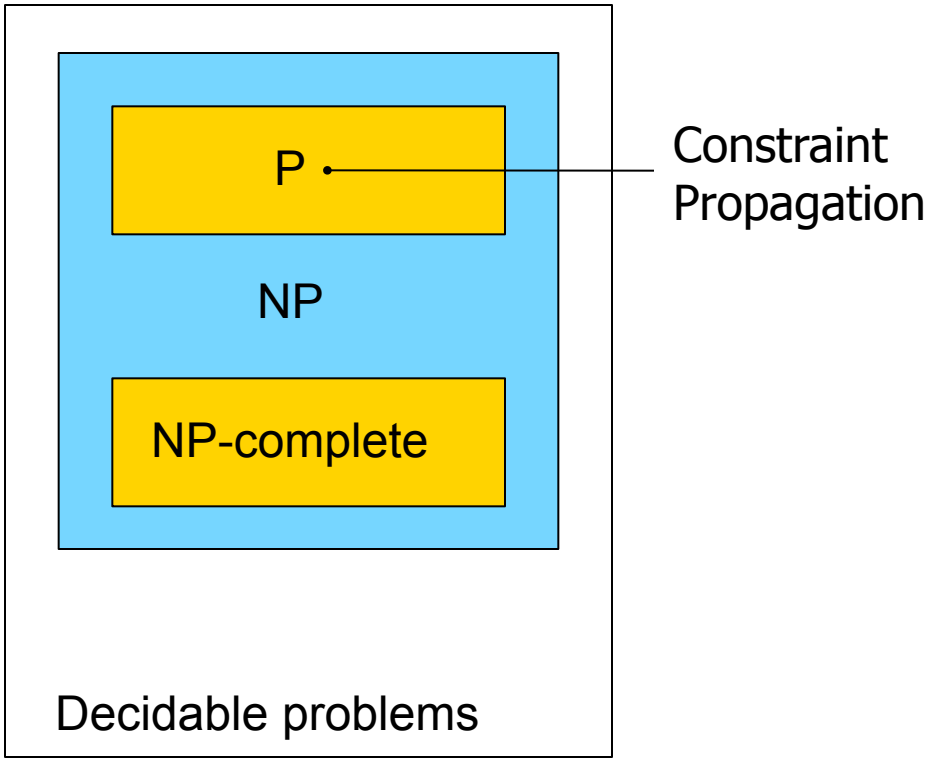
NP-Completeness

- Idea
  - NP-complete problems are hardest problems in NP
  - If efficient algorithm for one NP-hard problem would be found
    → P = NP.



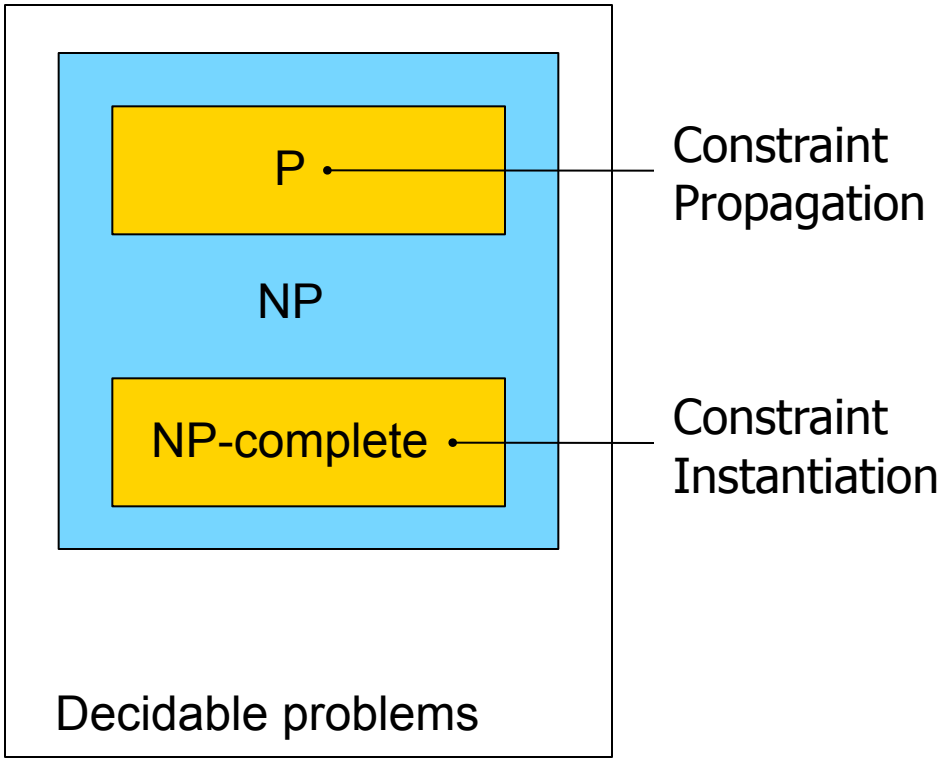
NP-Completeness

- Idea
  - NP-complete problems are hardest problems in NP
  - If efficient algorithm for one NP-hard problem would be found
    → P = NP.



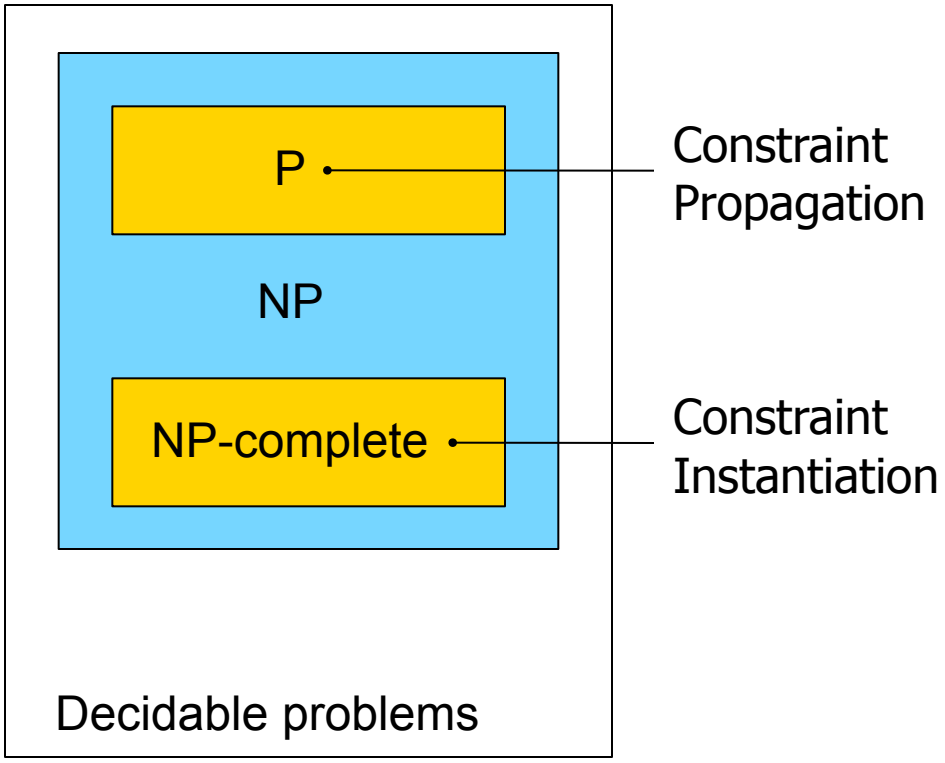
NP-Completeness

- Idea
  - NP-complete problems are hardest problems in NP
  - If efficient algorithm for one NP-hard problem would be found
    → P = NP.



NP-Completeness

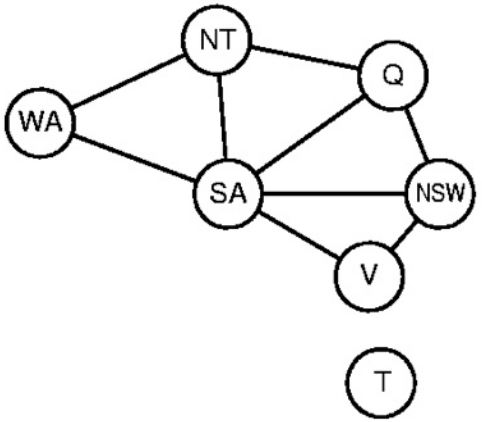
- Idea
  - NP-complete problems are hardest problems in NP
  - If efficient algorithm for one NP-hard problem would be found
    → P = NP.



Tackle NP-complete problems through polynomial preprocessing

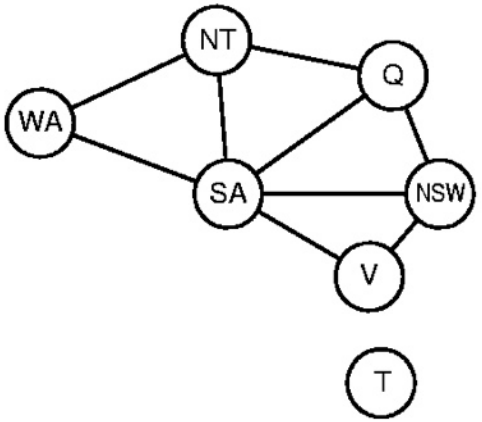
Forward Checking

- So far: Select-Unassigned-Variable
- Reduction of state space in a different way
  → Forward checking
- Forward checking (FC)
  - If variable X assigned, FC looks on all free variables that is connected to X



Forward Checking

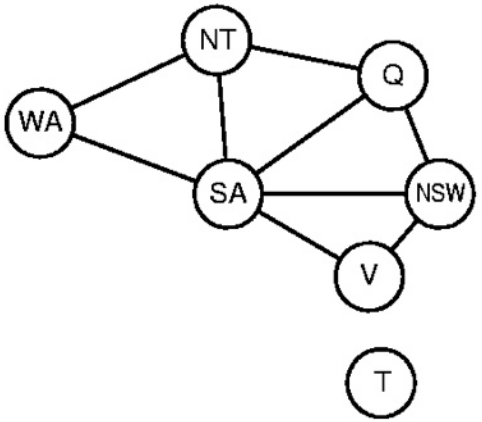
- So far: Select-Unassigned-Variable
- Reduction of state space in a different way
  → Forward checking
- Forward checking (FC)
  - If variable X assigned, FC looks on all free variables that is connected to X



	WA	NT	Q	NSW	V	SA	T
1. Begin	RGB	RGB	RGB	RGB	RGB	RGB	RGB

Forward Checking

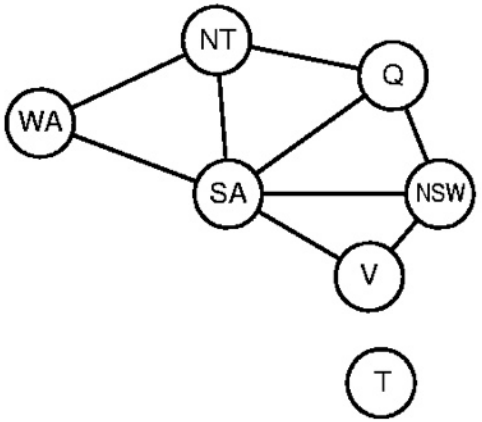
- So far: Select-Unassigned-Variable
- Reduction of state space in a different way
  → Forward checking
- Forward checking (FC)
  - If variable X assigned, FC looks on all free variables that is connected to X



	WA	NT	Q	NSW	V	SA	T
1. Begin	RGB	RGB	RGB	RGB	RGB	RGB	RGB
2. WA=red	R	GB	RGB	RGB	RGB	GB	RGB

Forward Checking

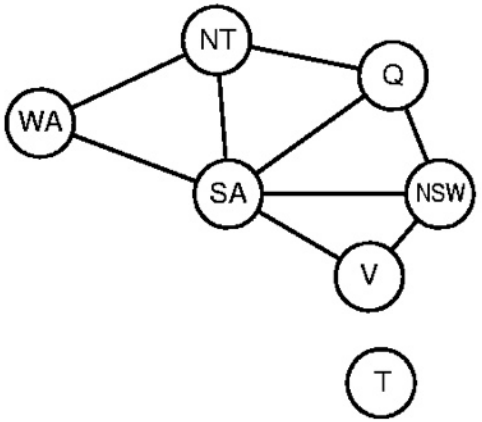
- So far: Select-Unassigned-Variable
- Reduction of state space in a different way
  → Forward checking
- Forward checking (FC)
  - If variable X assigned, FC looks on all free variables that is connected to X



	WA	NT	Q	NSW	V	SA	T
1. Begin	RGB	RGB	RGB	RGB	RGB	RGB	RGB
2. WA=red	R	GB	RGB	RGB	RGB	GB	RGB
3. Q=green	R	B	G	RB	RGB	B	RGB

Forward Checking

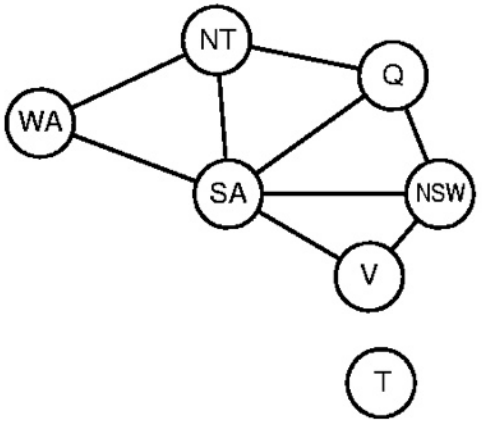
- So far: Select-Unassigned-Variable
- Reduction of state space in a different way
  → Forward checking
- Forward checking (FC)
  - If variable X assigned, FC looks on all free variables that is connected to X



	WA	NT	Q	NSW	V	SA	T
1. Begin	RGB	RGB	RGB	RGB	RGB	RGB	RGB
2. WA=red	R	GB	RGB	RGB	RGB	GB	RGB
3. Q=green	R	B	G	RB	RGB	B	RGB
4. V=blue	R	B	G	R	B		RGB

Forward Checking

- So far: Select-Unassigned-Variable
- Reduction of state space in a different way → Forward checking
- Forward checking (FC)
  - If variable X assigned, FC looks on all free variables that is connected to X

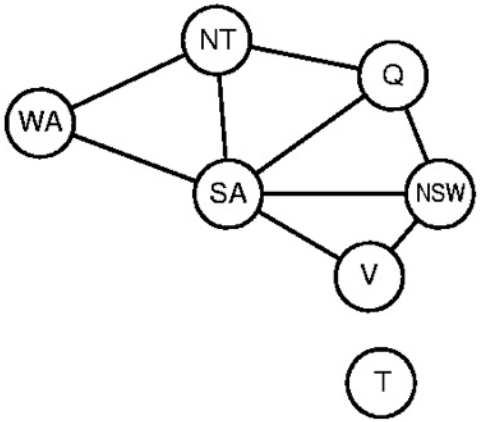


	WA	NT	Q	NSW	V	SA	T
1. Begin	RGB	RGB	RGB	RGB	RGB	RGB	RGB
2. WA=red	R	GB	RGB	RGB	RGB	GB	RGB
3. Q=green	R	B	G	RB	RGB	B	RGB
4. V=blue	R	B	G	R	B		RGB

- 3. NT and SA only one value: elimination of branching with information propagation, MRV selects NT and SA automatically
- 4. SA is empty → inconsistency detected

Constraint Propagation

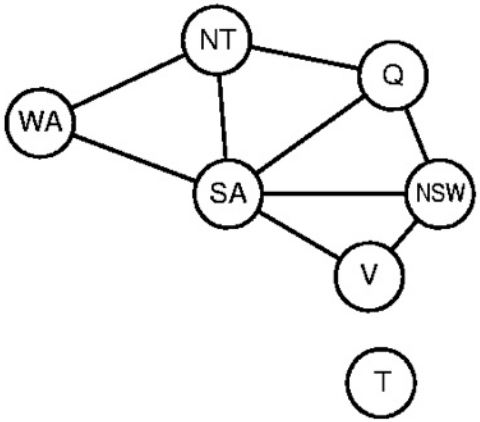
- Forward checking does not find all inconsistencies, e.g. 3. if WA=red & Q=green, there is only blue for NT and SA → neighbors!
- Forward checking does not predict far enough
- Constraint propagation is an expression for implications of a constraint from one variable to others
- Ex.: check NT- and SA-conditions in addition



	WA	NT	Q	NSW	V	SA	T
1. Begin	RGB	RGB	RGB	RGB	RGB	RGB	RGB
2. WA=red	R	GB	RGB	RGB	RGB	GB	RGB
3. Q=green	R	B	G	RB	RGB	B	RGB
4. V=blue	R	B	G	R	B		RGB

Arc-Consistency

- Fast method for constraint propagation
- Arc is directed, e.g. SA – NSW
- Arc is consistent if for every value x of SA there is at least one value y of NSW that is consistent with x (given: actual domains from SA and NSW
- 3. SA=blue, NSW={red,blue}. For SA=blue exists a consistent assignment NSW=red, thus, the arc SA-NSW is consistent
- Opposite: inconsistent: NSW=blue, no value for SA



	WA	NT	Q	NSW	V	SA	T
1. Begin	RGB	RGB	RGB	RGB	RGB	RGB	RGB
2. WA=red	R	GB	RGB	RGB	RGB	GB	RGB
3. Q=green	R	B	G	RB	RGB	B	RGB
4. V=blue	R	B	G	R	B		RGB

Arc-Consistency

- Arc consistency can be used
  - as preprocessing for search and
  - as propagation step (as FC)
- Repeat until inconsistencies are gone

- AC-3 as algorithm
- Time complexity $O(cd^3)$ *(with c binary constraints)*
- But: also this function does not find all possible inconsistencies

function AC3(*csp*) **returns** the CSP, possibly with reduced domains
  inputs: *csp*, a binary CSP with variables $\{X_1, X_2, \ldots, X_n\}$
  local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

  while *queue* is not empty **do**
    $(X_i, X_j) \leftarrow$ REMOVE-FRONT(*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) **returns** true iff we remove a value
  removed $\leftarrow$ *false*
  for each *x* **in** DOMAIN[X_i] **do**
    if no value *y* in DOMAIN[X_j] allows (*x*,*y*) to satisfy the constraint between X_i and X_j
      then delete *x* from DOMAIN[X_i]; *removed* $\leftarrow$ *true*
  end
  return *removed*

Special Constraints

- **Resource constraint** (also atmost constraint)
- Ex.: $P_1,\ldots,P_4$ is number of personnel that work on each of four tasks
- Constraint: no more than 10 personnel are assigned in total
  atmost(10,P_1,P_2,P_3,P_4)
- Inconsistency checking by checking the sum of the minimal values of the current domain, e.g. if each variable has the domain {3,4,5,6}, *atmost* cannot be satisfied
- Enforce consistency by deleting values, e.g. {2,3,4,5,6} →5,6 can be deleted to ensure consistency

- Big resource problems (e.g. logistics) almost impossible to keep domain as set of integers
- Upper and lower bounds
- **Bounds propagation** as solution
- Ex.: Two flights 271 and 272 with the capacities of 165 and 385 passengers, domains in the beginning:
  - Flight271 ∈ [0,165]
  - Flight272 ∈ [0,385]
- Constraint: both flights have to transport at least 420 passengers
  - Flight271 + Flight272 ∈ [420,420]
- With bounds propagation we get
  - Flight271 ∈ [35,165]
  - Flight272 ∈ [255,385]
- High relevance in practice!

Sudoku example

- Excellent example for CSP, people use CSP but might not know this.
- Board consists of 81 squares, some initially filled with 1-9.
- Puzzle is to fill all remaining digits with constraints.
- Exactly ONE solution.
- Puzzle can be considered a CSP with 81 variables, one for each square. We use the variable names A1 through A9 for the top row (left to right), down to I1 through I9 for the bottom row.

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

(a)

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

(b)

Sudoko example (2)

- Empty squares have the domain {1, 2, 3, 4, 5, 6, 7, 8, 9}
- 27 different *Alldiff* constraints: one for each row, column, and box of 9 squares.

$$
\begin{aligned}
&\mathit{Alldiff}(A1, A2, A3, A4, A5, A6, A7, A8, A9)\\
&\mathit{Alldiff}(B1, B2, B3, B4, B5, B6, B7, B8, B9)\\
&\ldots\\
&\mathit{Alldiff}(A1, B1, C1, D1, E1, F1, G1, H1, I1)\\
&\mathit{Alldiff}(A2, B2, C2, D2, E2, F2, G2, H2, I2)\\
&\ldots\\
&\mathit{Alldiff}(A1, A2, A3, B1, B2, B3, C1, C2, C3)\\
&\mathit{Alldiff}(A4, A5, A6, B4, B5, B6, C4, C5, C6)\\
&\ldots
\end{aligned}
$$

Sudoku example (3)

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

(a)

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

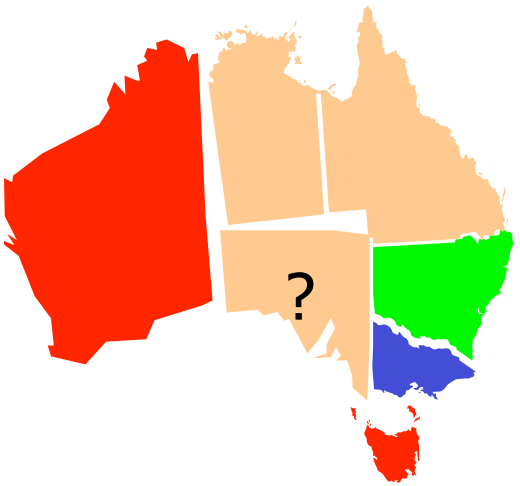
(b)

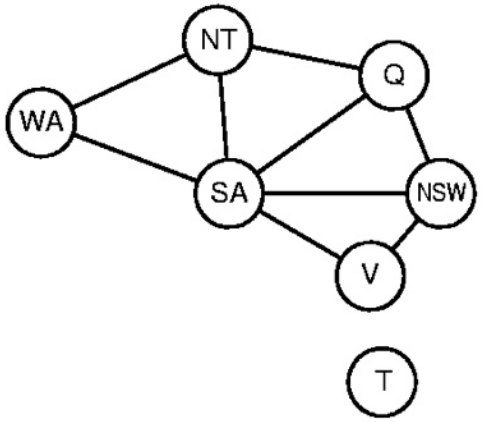
Remarks

- AC-3 works only for the easiest Sudoku puzzles.
- Slightly harder ones can be solved by PC-2.
- Larger computational cost: there are 255,960 different path constraints to consider.
- Solving the hardest puzzles and to make efficient progress, more intelligence is needed.

Intelligent Backtracking

- BT also called chronological BT
- Takes last marked variable
- Better: intelligent selection
- Ex.: Fixed order Q, NSW, V, T, SA, WA, NT, partial assignment {Q=red, NSW=green,V=blue,T=red}
- Next variable SA: problem!
- Backtracking to T and new assignment → not a good choice
- Better: go back to set of variable that caused the problem →conflict set
- Here: conflict set for SA is {Q,NSW,V}





- Backjumping instead of Backtracking

Local Search: min-conflict heuristic

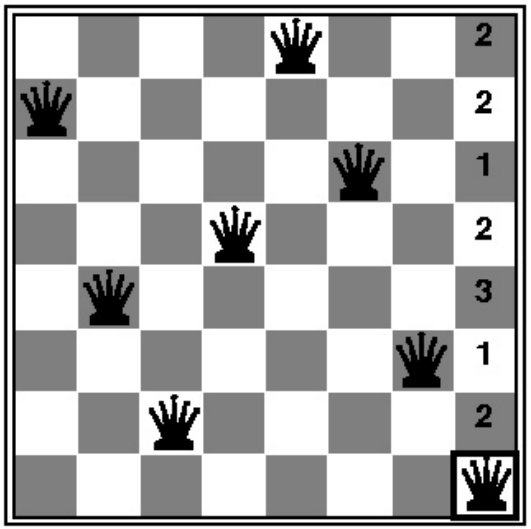
- Local search effective with CSPs
- Heuristic: select value that minimizes the number of conflicts

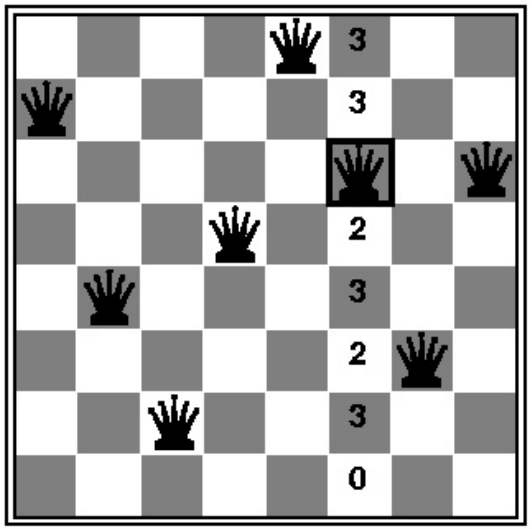
- Very effective, n-queens problem in less than 50 steps, also with 1 million queens
- Relevant for practice

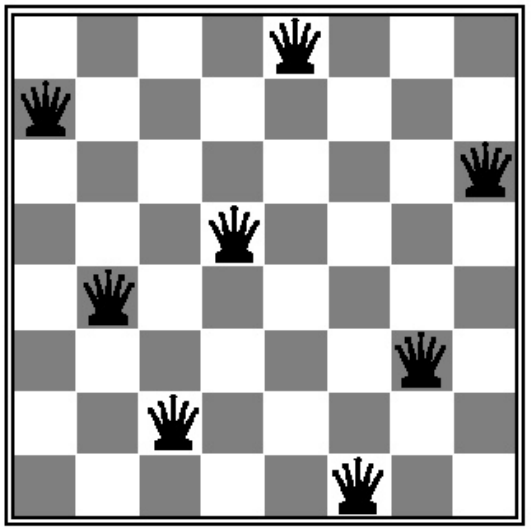
```
function MIN-CONFLICTS(csp, max-steps) returns a solution or failure
  inputs: csp, a constraint satisfaction problem
          max-steps, the number of steps allowed before giving up
  local variables: current, a complete assignment
                   var, a variable
                   value, a value for a variable

  current ← an initial complete assignment for csp
  for i = 1 to max-steps do
      if current is a solution for csp then return current
      var ← a randomly chosen, conflicted variable from VARIABLES[csp]
      value ← the value v for var that minimizes CONFLICTS(var, v, current, csp)
      set var = value in current
  return failure
```

Example: Min-conflict Heuristic







- Solution for 8-queens problem in 2 steps
- Each step, one queen is selected for new assignment
- Shown: number of conflicts

- Algorithm follows minimal conflicts
- > 1 minimum: random selection

Comparison of CSP-Algorithms

Problem	BT	BT+MRV	FC	FC+MRV	Min-Conflicts
USA	(> 1,000K)	(> 1,000K)	2K	60	64
n-Queens	(> 40,000K)	13,500K	(> 40,000K)	817K	4K
Zebra	3,859K	1K	35K	0.5K	2K
Random 1	415K	3K	26K	2K	
Random 2	942K	27K	77K	15K	

- Algorithms
  - Simple Backtracking
  - Backtracking with MRV Heuristic
  - Forward checking
  - Forward checking with MRV Heuristic
  - Minimum conflicts locale search
- Problems
  - Color mapping for USA
  - Number of checks to solve n-queens problem for (n=2-50)
  - Zebra-Puzzle
  - Two artificial problems

- Median over consistency checking (5 runs) for problem solution
- () = no solution found within number of checks
- Result:
  - FC with MRV better than BT algorithms
  - Does not always hold for Min-Conflicts

Structure of problems

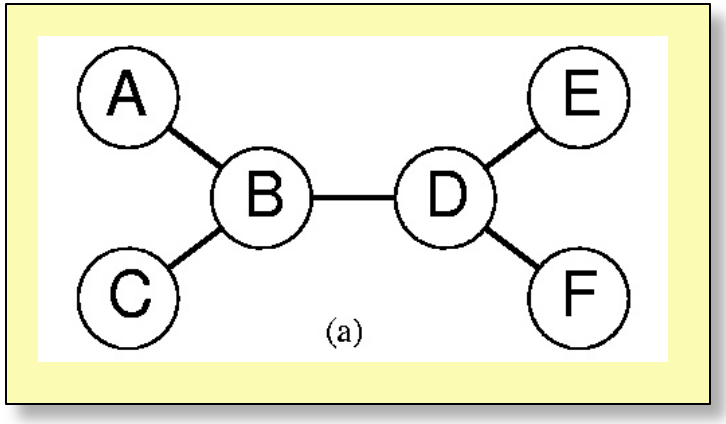
- Independent sub-problems (e.g. Tasmania and main land)
- Can be achieved by looking for components that have connections in graph
- Component = sub-problem
- If assignment S_i solution of sub-problem, $\cup_i S_i$ is a solution for $\cup_i CSP_i$
- Why is this important?

- Assumption: CSP_i has *c* variables of a total of *n*, *c* is a constant, then there are *n/c* sub-problems with minimal d^c time for the solution
- In total: $O(d^c\, n/c)$, i.e. linear in *n*
- Without decomposition $O(d^n)$, i.e. exponential
- Concrete: a decomposition of a boolean CSP with *n=80* in four sub-problems with *c=20* reduces the problem in the worst-case from lifetime of universe to less than 1 second!

Structure of problems

- Mostly, sub-problems in CSPs not independent
- Simple case: CG forms a tree

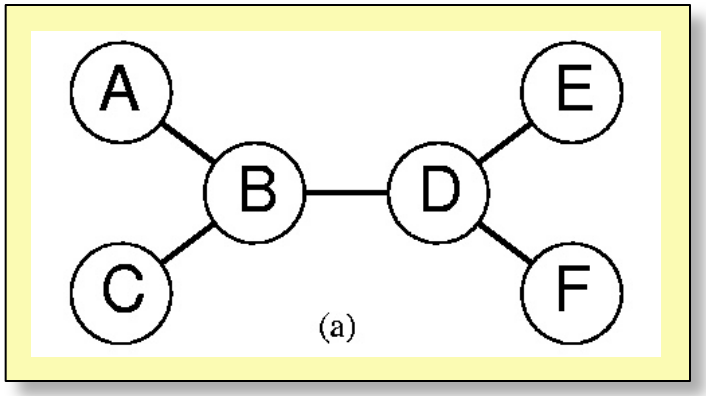
- Time complexity for tree-like CSPs is linear in *n*

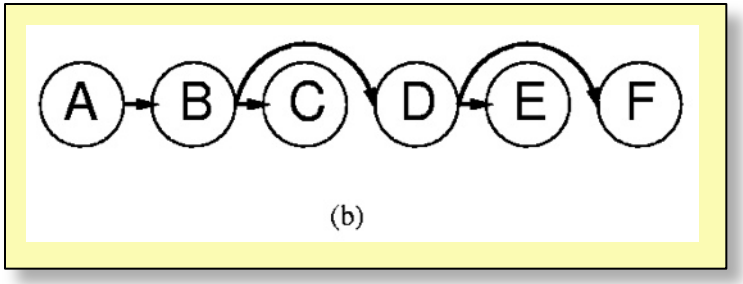


Structure of problems

Algorithm: $O(nd^2)$

- **Select root arbitrary and order all variables such that the parents of a node are in front of the children (b)**
- **For j from n down to 2**
  arc-consistency (X_i, X_j),
  with X_i = parent node of X_j
- **For j from 1 to n**
  Value assignment all values to X_j that are consistent with value from X_i, with X_i = parent node of X_j

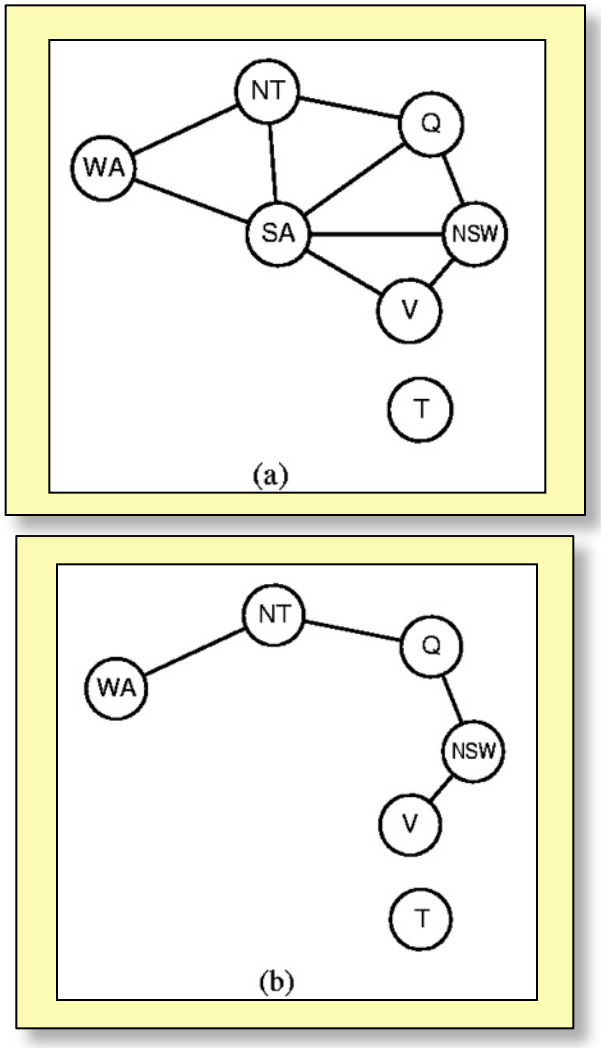




- This knowledge: try to reduce general problems to trees. Two approaches:
  - Cutset conditioning
  - Tree decomposition

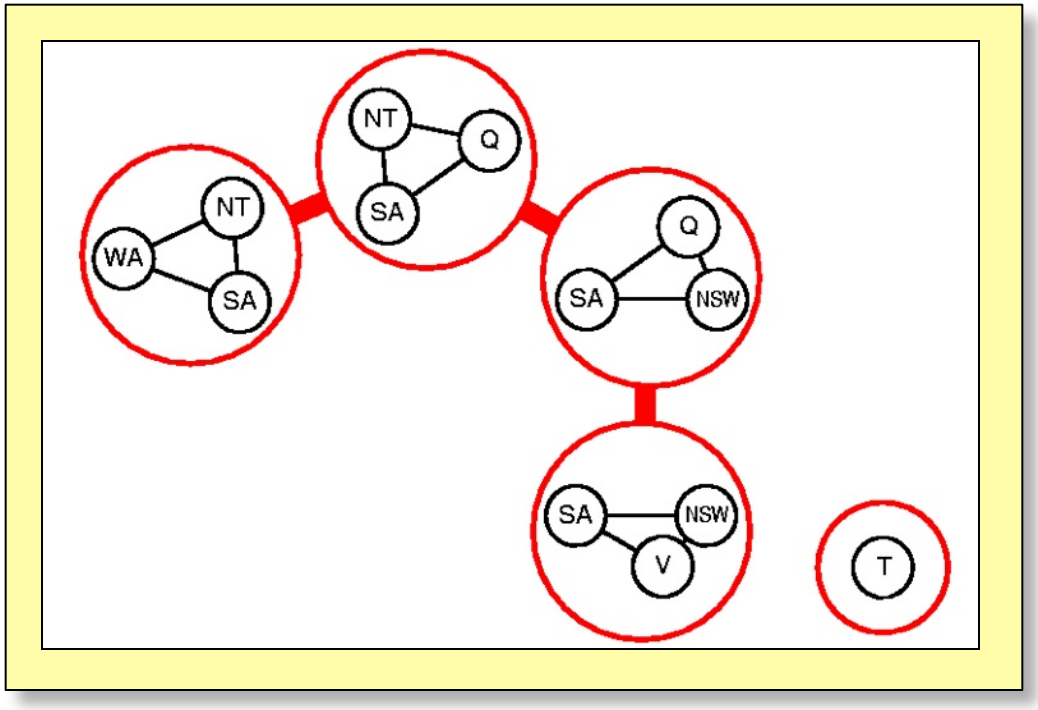
Cutset conditioning

- Ex.: Elimination of SA, fixed value, value will be deleted for all variables
- We get a tree
- "Cutset", because algorithm selects a subset S, so that CG becomes a tree, S also labeled "cycle cutset"



Tree decomposition

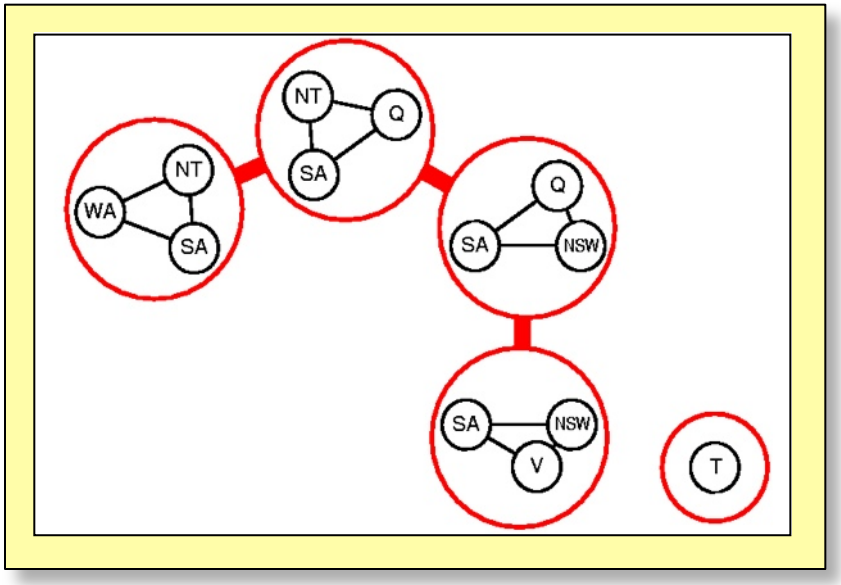
- Idea: decompose CG in subsets, which are connected
- Each sub-problem will be solved separately and then results are combined
- Works well as long as sub-problems get not too big (like divide-and-conquer)
- Here: five sub-problems



- Conditions
  - Every variable in the original problem appears in at least one of the sub-problems.
  - If two variables are connected by a constraint in the original problem, they must appear together (along with the constraint) in at least one of the sub-problems.
  - If a variable appears in two sub-problems in the tree, it must appear in every sub-problem
  - along the path connecting those sub-problems.

Tree decomposition

- Width of a tree defined as size of largest sub-problem-1
- Width of CG defined through minimal tree of all tree decompositions
- When graph has width *w* and we have corresponding tree decompositions, we can solve the problem in $O(nd^{w+1})$
- → **CSPs with constraint graphs of bounded tree width are solvable in polynomial time!**



Summary

Summary

- **Constraint satisfaction problems** (or CSPs) consist of variables with constraints on them. Many important, real-world problems can be described as CSPs. The structure of a CSP can be represented by its constraint graph.

Summary

- **Constraint satisfaction problems** (or CSPs) consist of variables with constraints on them. Many important, real-world problems can be described as CSPs. The structure of a CSP can be represented by its constraint graph.
- **Backtracking** search, a form of depth-first search, is commonly used for solving CSPs.

Summary

- **Constraint satisfaction problems** (or CSPs) consist of variables with constraints on them. Many important, real-world problems can be described as CSPs. The structure of a CSP can be represented by its constraint graph.
- **Backtracking** search, a form of depth-first search, is commonly used for solving CSPs.
- The **minimum remaining values** and **degree** heuristics are domain-independent methods for deciding which variable to choose next in a backtracking search. The **least-constraining-value** heuristic helps in ordering the variable values.

Summary

- **Constraint satisfaction problems** (or CSPs) consist of variables with constraints on them. Many important, real-world problems can be described as CSPs. The structure of a CSP can be represented by its constraint graph.
- **Backtracking** search, a form of depth-first search, is commonly used for solving CSPs.
- The **minimum remaining values** and **degree** heuristics are domain-independent methods for deciding which variable to choose next in a backtracking search. The **least-constraining-value** heuristic helps in ordering the variable values.
- By propagating the consequences of the partial assignments that it constructs, the back-tracking algorithm can reduce greatly the branching factor of the problem. **Forward checking** is the simplest method for doing this. **Arc consistency** enforcement is a more powerful technique, but can be more expensive to run.

Summary

Summary

- Backtracking occurs when no legal assignment can be found for a variable. **Conflict-directed backjumping** records the variables contributing to the failure and backtracks to the most recent one.

Summary

- Backtracking occurs when no legal assignment can be found for a variable. **Conflict-directed backjumping** records the variables contributing to the failure and backtracks to the most recent one.
- Local search using the **min-conflicts** heuristic has been applied to constraint satisfaction problems with great success.

Summary

- Backtracking occurs when no legal assignment can be found for a variable. **Conflict-directed backjumping** records the variables contributing to the failure and backtracks to the most recent one.
- Local search using the **min-conflicts** heuristic has been applied to constraint satisfaction problems with great success.
- The complexity of solving a CSP is strongly related to the structure of its constraint graph. Tree-structured problems can be solved in linear time. **Cutset conditioning** can reduce a general CSP to a tree-structured one, and is very efficient if a small cutset can be found. Tree decomposition techniques transform the CSP into a tree of sub-problems, and are efficient if the tree width of the constraint graph is small.