Lecture 10: Search 9

Victor R. Lesser

CMPSCI 683

Fall 2010

Announcement

- Office Hours Today from 1:30-2:45
- Hala will send out a new version of the simulator fixing some minor problems
- Exam on Monday October 18; covering all material including *today's and tomorrow's* lecture
  - Open book but *not access to internet*
  - Required readings but not optional readings

GSAT Algorithm

Problem: Given a formula of the propositional calculus, find an interpretation of the variables under which the formula comes out true, or report that none exists.

procedure GSAT

Input: a set of clauses $\propto$, MAX-FLIPS, and MAX-TRIES

Output: a satisfying truth assignments of $\propto$, if found

begin

for ***i***:= 1 **to** MAX-TRIES *; random restart mechanism*

T := a randomly generated truth assignment

for *j* := 1 **to** MAX-FLIPS

if *T* satisfies $\propto$ **then return** *T*

p := a propositional variable such that a change in its truth assignment gives the largest increase in total number of clauses of $\propto$ that are satisfied by *T*. *; needs to be highly efficient*

T := *T* with the truth assignment of *p* reversed

end for

end for

return "no satisfying assignment found"

end

3SAT Phase Transition

- Easy -- Sastifiable problems where many solutions
- **Hard -- Sastifiable problems where few solutions**
- Easy -- Few Satisfiable problems

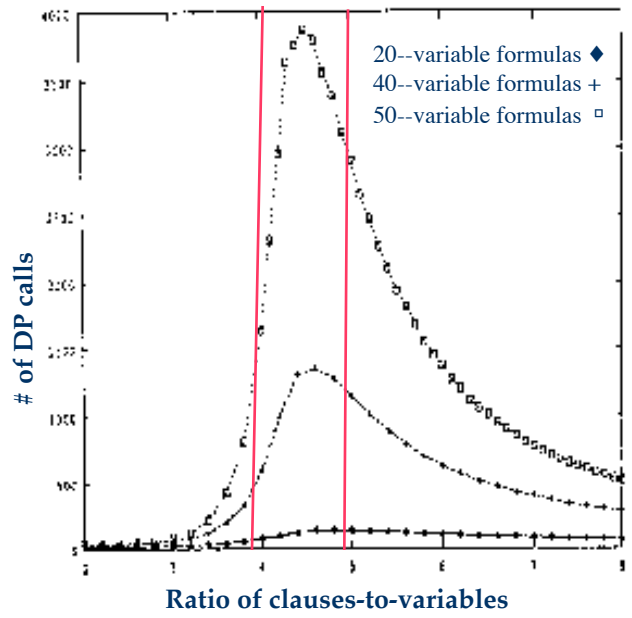


Fig. 1 Solving 3SAT problems.

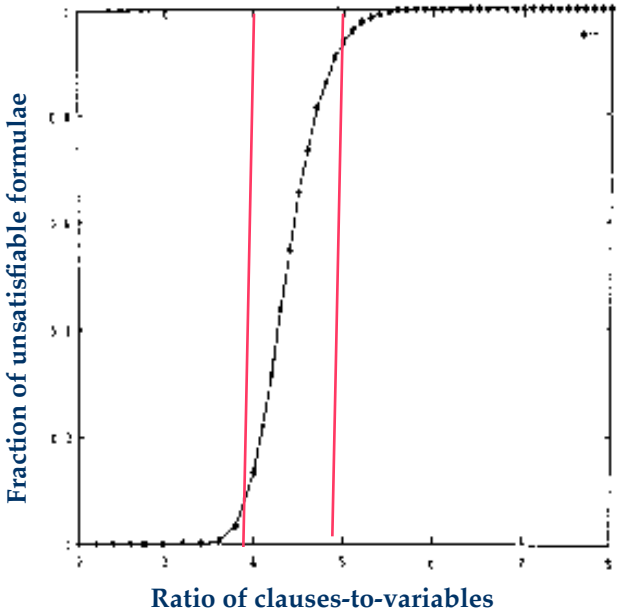


Fig. 2 Fraction of unsatisfiable 3SAT problems.

- Assumes concurrent search in the satisfiable space and the non-satisfiable space (negation of proposition)

This Lecture

- Informed-Backtracking Using Min-Conflicts Heuristic
- Arc Consistency for Pre-processing
- Intelligent backtracking
- Reducing the Search by structuring the CSP as a tree search

Depth-First CSP Search with Single-Variable Assignments -- Backtracking Search

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
    return RECURSIVE-BACKTRACKING([], csp)   ; starts out with no variable/value in assignments

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
            result ← RECURSIVE-BACKTRACKING([var = value—assignment], csp)   ; Expand depth-first
            if result ≠ failure then return result
    end
    return failure
```

Recursion implicitly holds the search tree and the possibilities that have not been explored

Intelligent Search for CSPs

- CSP search complexity may be affected by:
  - The order in which variables are assigned values;
  - The domain values chosen for assignment.
- Variable-ordering heuristics reduce the bushiness of the search tree by moving failures to upper levels.
- Value-ordering heuristics move solutions to the “left” of the search tree so they are found more quickly by backtracking search.
- Good heuristics can reduce search complexity by nearly an order of magnitude.

Informed-Backtracking Using Min-Conflicts Heuristic

```
Procedure INFORMED-BACKTRACK (VARS-LEFT VARS-DONE)
    If all variables are consistent, then solution found, STOP.
    Let VAR = a variable in VARS-LEFT that is in conflict.; HOW TO CHOOSE?
    Remove VAR from VARS-LEFT.
    Push VAR onto VARS-DONE.
    Let VALUES = list of possible values for VAR ordered in ascending
                 order according to number of conflicts with variables
                 in VARS-LEFT. – min-conflict heuristic
    For each VALUE in VALUES, until solution found:
        If VALUE does not conflict with any variable that is in VARS-DONE,
        then Assign VALUE to VAR.
                 Call INFORMED-BACKTRACK(VARS-LEFT VARS-DONE); DEPTH-FIRST RECURSION
        end if
    end for; WHAT HAPPENS IF YOU DON'T FIND ACCEPTABLE VALUE?
end procedure
Begin program (INITIALIZATION OF RECURSIVE BACKTRACKING)
    Let VARS-LEFT = list of all variables, each assigned an initial state
    Let VARS-DONE = nil
    Call INFORMED-BACKTRACK(VARS-LEFT VARS-DONE)
End program
```

; start with with all variables in vars-left

Heuristics that can help

Key questions:

1. Which variable should be assigned next and in what order should the values be tried?
2. What are the implications of the current variable assignments for the other unassigned variables?
3. When a path fails, can the search avoid repeating this failure in subsequent paths?

Number of backtracks/repairs for *N*-Queens algorithms *(S. Minton et al.)*

	Constructive		Repair-based	
n	Standard backtrack	Most constrained backtrack¨	Min-conflicts hill-climbing	Min-conflicts backtrack
$n = 10^1$	53.8	17.4	57.0	46.8
$n = 10^2$	4473 (70%)	687 (96%)	55.6	25.0
$n = 10^3$	88650 (13%)	22150 (81%)	48.8	30.7
$n = 10^4$	*	*	48.5	27.5
$n = 10^5$	*	*	52.8	27.8
$n = 10^6$	*	*	48.3	26.4

* = exceeded computation resources

Potential Reasons for Heuristic Repair to be Advantageous

- Depth-first search badly organized
  - Poorer choices are explored first at each branch point
  - More solutions with first queen placed in center of first row
    - *Takes a very long time to recover from bad decision made early in search*
  - Backtracking program that randomly orders rows (and columns within rows) still performs poorly
- Distribution of solutions
  - Depth first does not perform well where solutions clustered in tree
  - Random backtracking (Las Vegas algorithm) does better but still problem

Potential Reasons for Heuristic Repair to be Advantageous *(cont'd)*

- Informedness hypothesis
  - Heuristic repair is better because it has more information that is not available to a constructive backtracking (more encompassing view of search space)
  - Mini-conflict heuristic — select a variable that is in conflict and assign it a value that minimizes the number of conflicts (number of other variables that will need to be repaired)

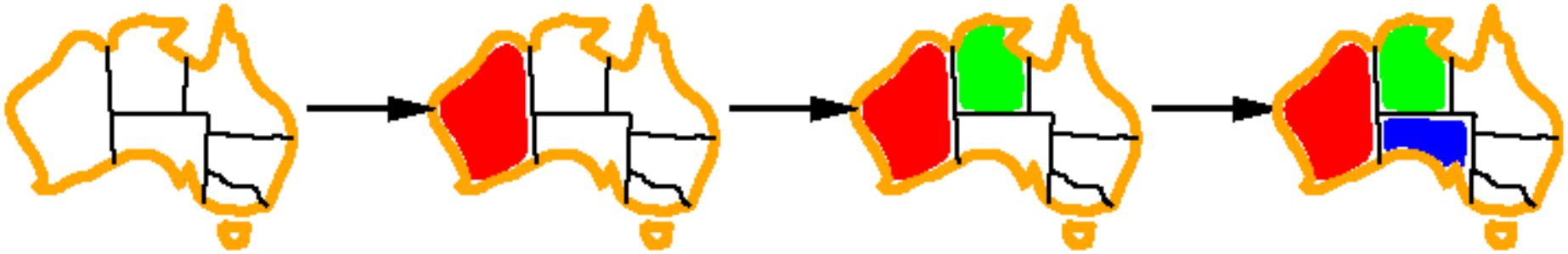
Other Examples of Heuristics for CSPs

- Most-constraining variable
  - Select for assignment the variable that is involved in the largest number of constraints on unassigned variables;
  - Also called the *search-rearrangement method*;

- Least-constraining value
  - Select a value for the variable that eliminates the smallest number of values for variables connected with the variable by constraints;
  - i.e., maximize the number of assignment options still open.

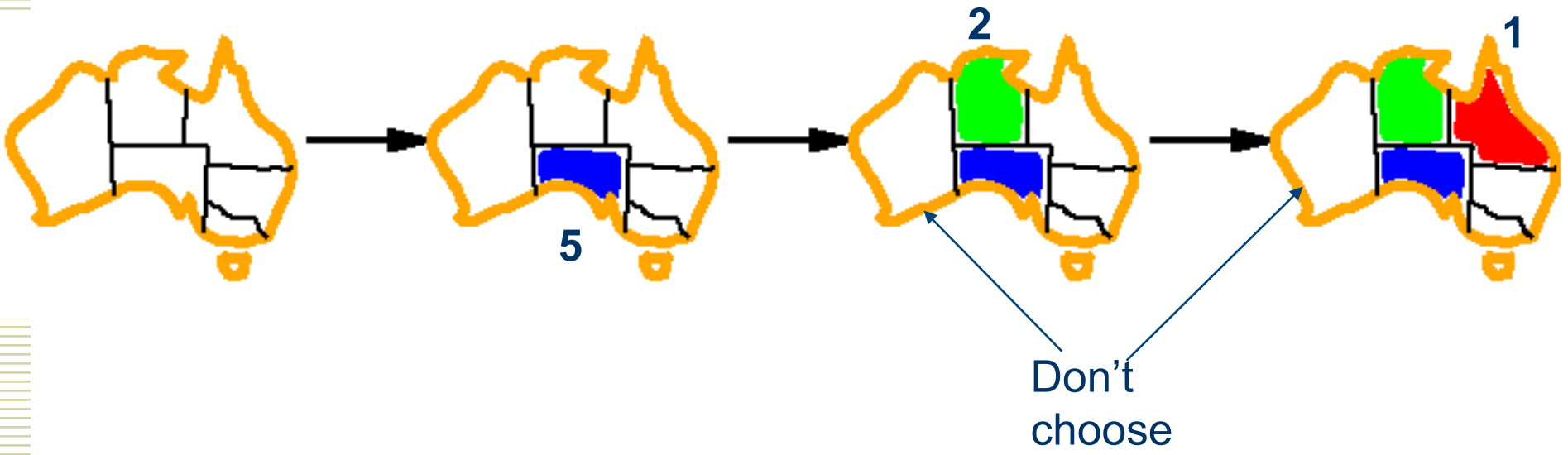
Most constrained variable

Choose the variable with the fewest legal values



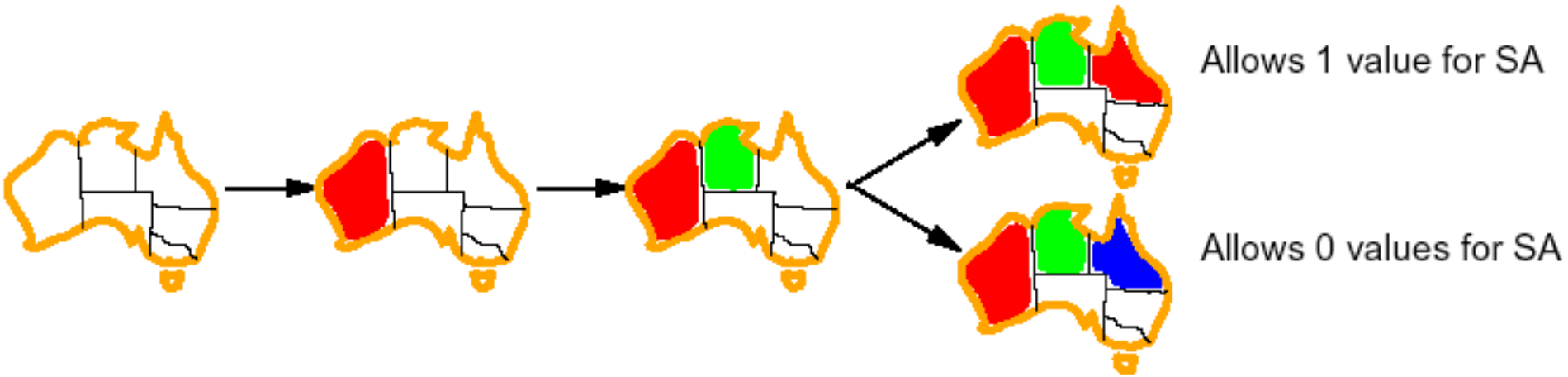
Most constraining variable

- Tie-breaker most constrained variables
- Choose the variable with the most constraints on remaining variables



Least constraining value

- Choose the one that rules out the fewest values in the remaining variables



- Combining all these heuristics, make 1000 queens feasible

Some Additional Ideas on CSP Search

- Arc Consistency for Pre-processing
- Other approaches to ordering variables and values in search

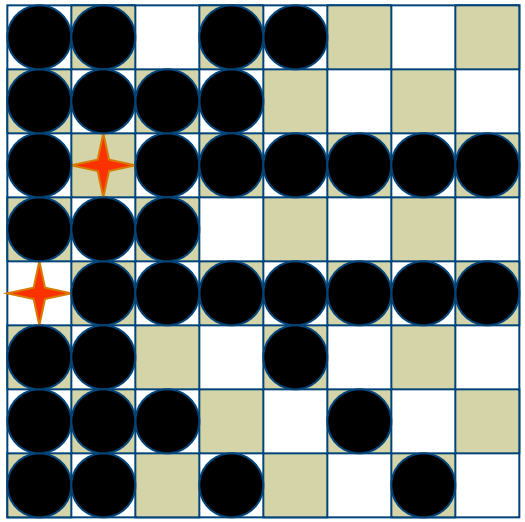
Heuristics that can help

Key questions:

1. Which variable should be assigned next and in what order should the values be tried?
2. What are the implications of the current variable assignments for the other unassigned variables?
3. When a path fails, can the search avoid repeating this failure in subsequent paths?

Constraint propagation …

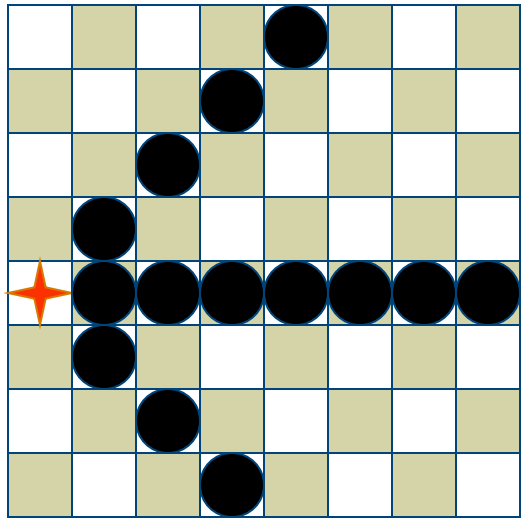
… is the process of determining how the possible values of one variable affect the possible values of other variables



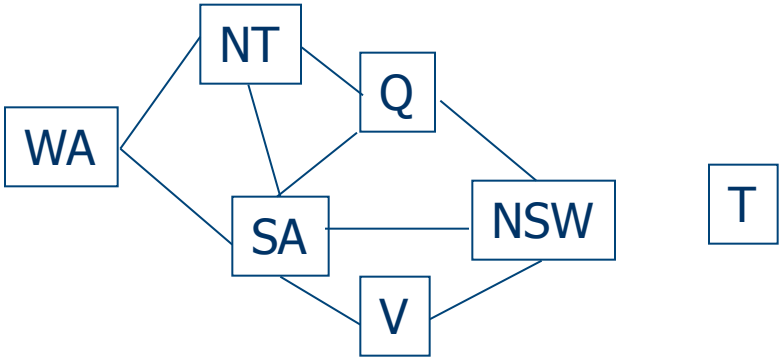
The placement of the two queens makes the placement of queens in the black dots invalid

Forward Checking: A Simple kind of Propagation

- After a variable X is assigned a value v, look at each unassigned variable Y that is connected to X by a constraint and deletes from Y's domain any value that is inconsistent with v
- Reduces the branching factor and help identify failures early.

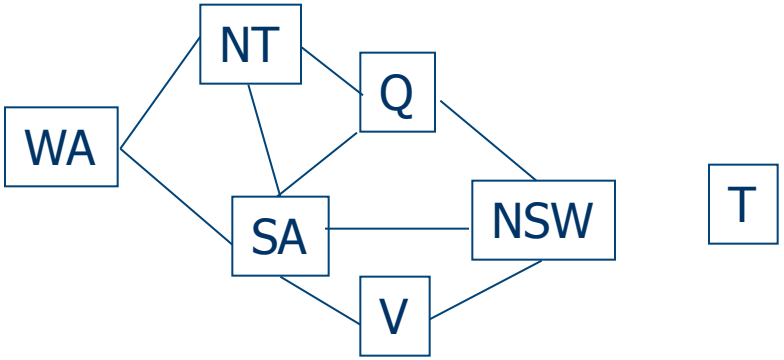


Map coloring



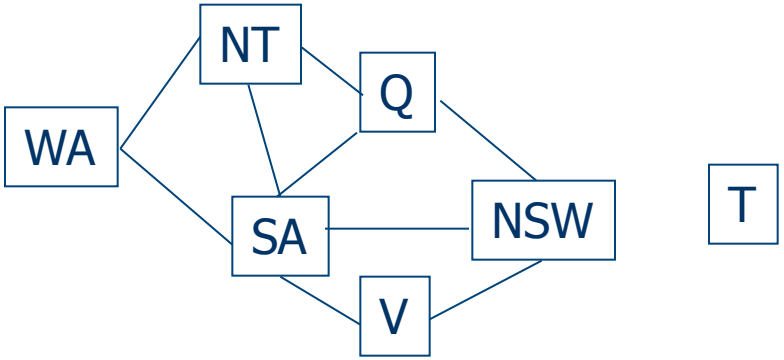
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB

Map coloring



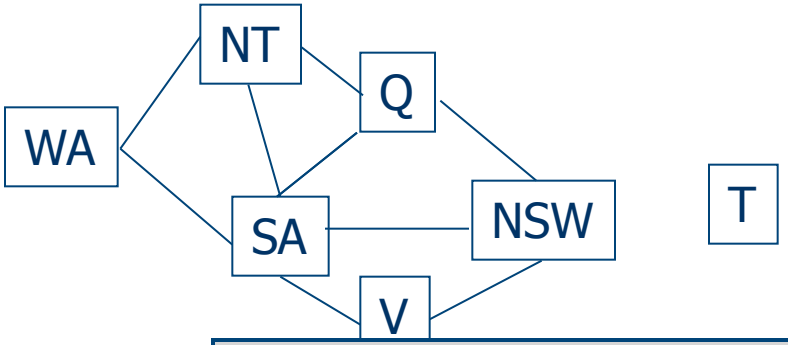
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB

Map coloring



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	B	G	RB	RGB	B	RGB

Map coloring



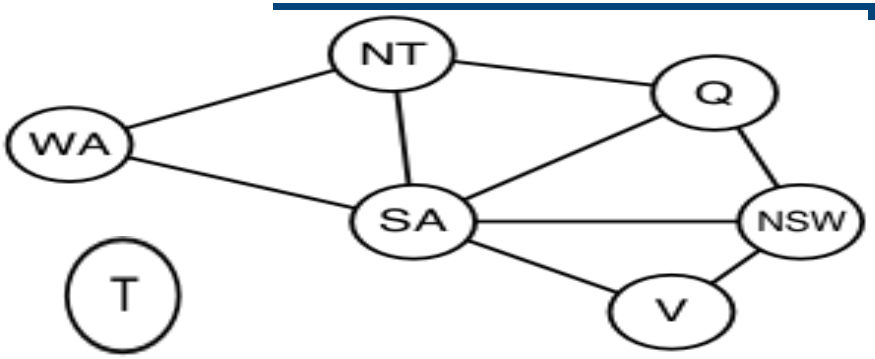
Impossible assignments that forward checking does not detect

WA	NT	Q				
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	B	G	RB	RGB	B	RGB
R	B	G	R	B		RGB

Arc (K=2) consistency

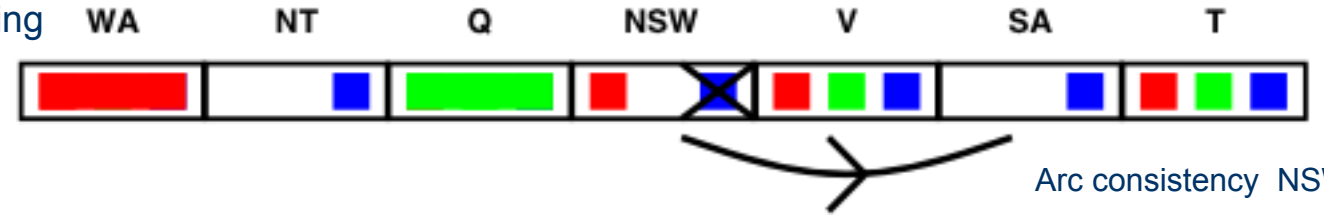
- An arc from *X* to *Y* in the constraint graph is consistent if, for *every* value of *X*, there is *some value of Y* that is consistent with *X*.
- *Can detect more inconsistencies than forward checking.*
  - Can be applied as a preprocessing step before search
  - As a propagation step after each assignment during search. -- how is this advantageous
- Process must be applied *repeatedly* until no more inconsistencies remain. Why?

ARC Consistency Example

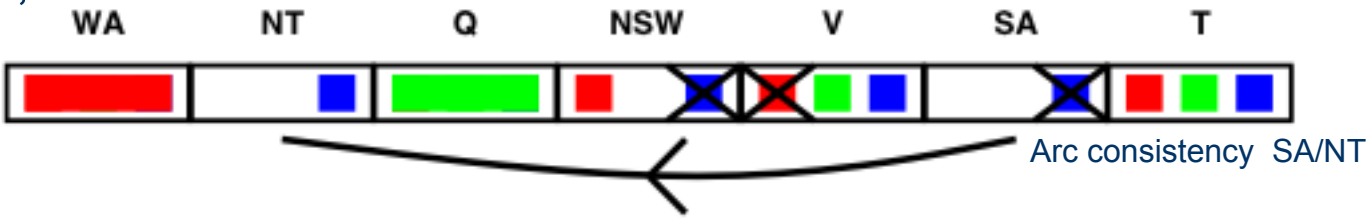


For NSW=B is there any valid set of its neighbors that will allow its constraints to be satisfied; then for V= R, and finally SA=B

Forward checking (WA,Q)



Update V/NSW,



No possible solution with WA=red and Q=green

ARC Consistency Algorithm

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*

  loop while *queue* is not empty **do** ; propagates effects thru network
    $(X_i, X_j) \leftarrow$ 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*
end

; If X_i has an inconsistent domain value then need to reassess relations with other neighbors

---

function REMOVE-INCONSISTENT-VALUES(X_i, X_j) **returns** true iff we remove a value
  removed $\leftarrow$ *false*
  loop for each x **in** DOMAIN[X_i] **do**
   If (x,y) arc consistency can not be satisfied with some value y in DOMAIN[X_j] then delete x from DOMAIN[X_j]; remove<-true
  end
  return *removed*
end

Complexity of arc consistency

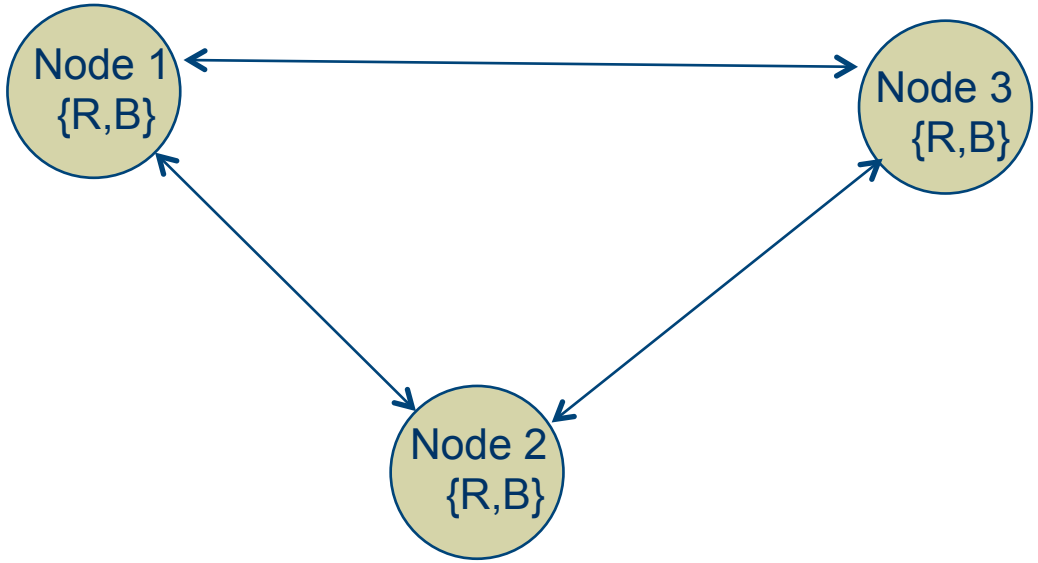
- A binary CSP has at most O(n^2) arcs
- Each arc ($X \rightarrow Y$) can only be inserted on the agenda d times because at most d values of Y can be deleted.
- Checking consistency of an arc can be done in O(d^2) time.
- Worst case time complexity is: O($n^2 d^3$).
- **Does not reveal every possible inconsistency**!
  - Does not take into account multiple constraints simultaneously
  - The only value of X than can satisfy the constraint from Y is 5 while the only of X than can satisfy the constraint form W is 6; in this case there is no solution

K-consistency

- A graph is *k*-consistent if, for any set of *k* variables, there is always a consistent value for the *k*th variable given any consistent partial assignment for the other *k*-1 variables.
  - A graph is strongly k-consistent if it is *i*-consistent for $i = 1..k$.
  - IF k=number of nodes than no backtracking
- Higher forms of consistency offer stronger forms of constraint propagation.
  - Reduce amount of backtracking
  - Reduce effective branching factor
  - Detecting inconsistent partial assignments
- Balance of how much pre-processing to get graph to be k consistent versus more search

Need for 3-Consistency

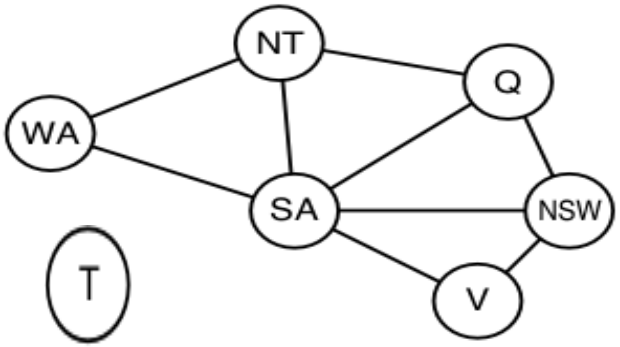
- With 2-Consistency no further reduction



- With 3-Consistency realizes problem is unsolvable: ({N1=R,N2=B},{N1=B, N2=R}) no consistent assignment for node 3

Intelligent Backtracking: BackJumping

- **Chronological backtracking**: always backtrack to most recent assignment. Not efficient!
- **Conflict set**: A set of variables that caused the failure.
- **Backjumping**: backtrack to the most recent variable assignment in the conflict set.
- Simple modification of BACKTRACKING-SEARCH.



Fixed variable ordering Q,NSW,V,T,SA,WA,NT

{Q,NSW,V,T}, SA=?; backup to T makes no sense

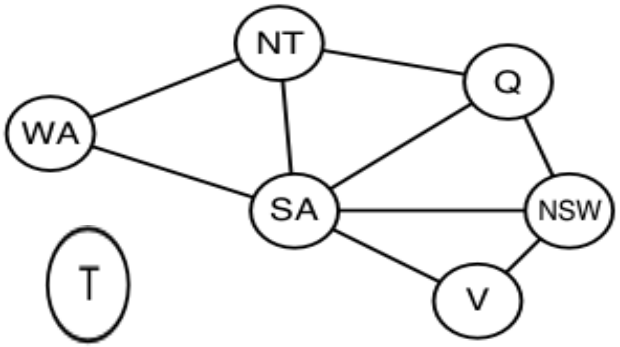
What Variable(s) Caused the Conflict

Backtrack to V, most recent variable set in conflict set

- **Forward Checking can be used to generate conflict set based on variables that remove elements from domain of other variables**

More Advanced Backtracking

- **Conflict-directed backjumping**: better definition of conflict sets leads to better performance -- ***bottom-up/top-down state integration***



WA=red, NSW=red can never be solved

T= red, then assign NT,Q,V,SA (always fails)

How to know that (indirect) conflict set of NT is WA and NSW since they don't conflict with NT

Conflict set of NT is set of preceding variables that caused NT, together with any subsequent variables, to have no consistent solutions

SA fails conflict {WA,NT,Q} based on forward propagation; backjump to Q

Q absorbs conflict set of SA minus Q {WA,NSW,NT}; backjump to NT

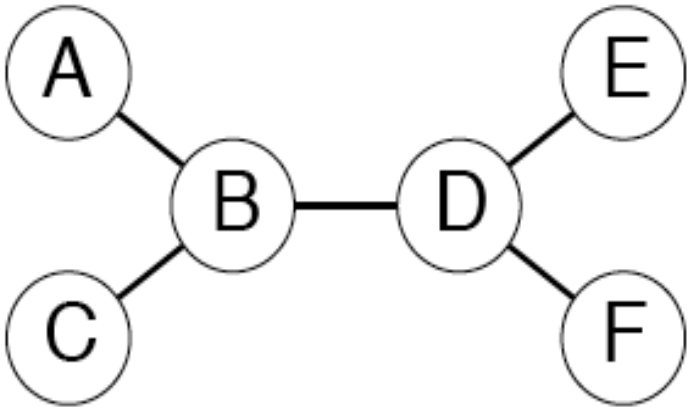
NT absorbs conflict set of Q minus NT {WA,NSW};

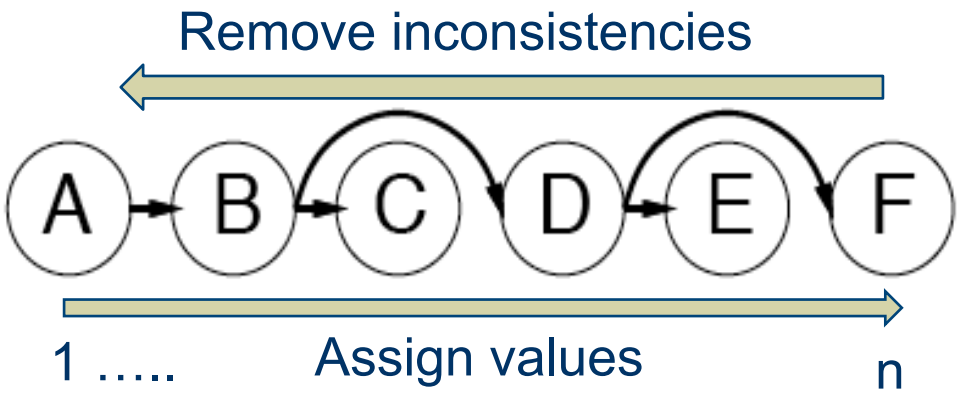
Complexity and problem structure

- The complexity of solving a CSP is strongly related to the **structure** of its constraint graph.
- Decomposition into **independent subproblems** yields substantial savings: $O(d^n) \rightarrow O(d^c \cdot n/c)$
- **Tree-structured problems** can be solved in linear time $O(n \cdot d^2)$
- **Cutset conditioning** can reduce a general CSP to a tree-structured one, and is very efficient if a small cutset can be found.

Algorithm for Tree Structured CSPs

1. Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering



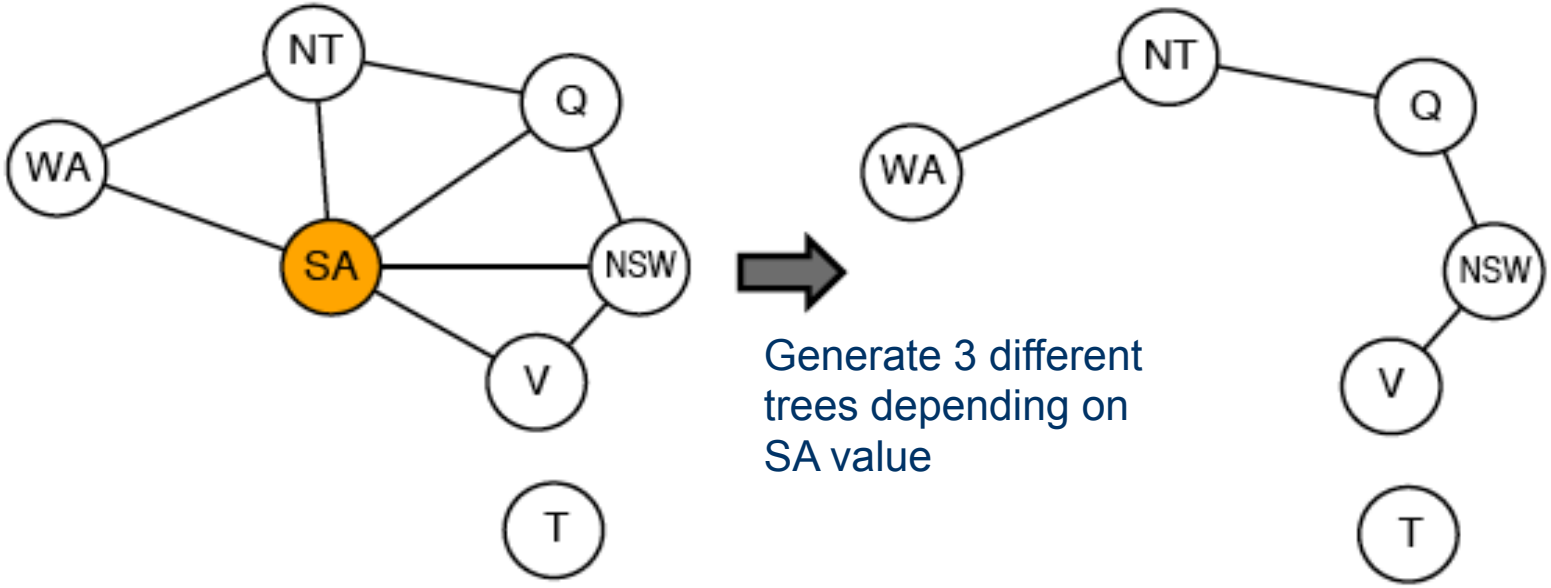


2. For j from n down to 2, apply $\text{REMOVEINCONSISTENT}(Parent(X_j), X_j)$

3. For j from 1 to n, assign X_j consistently with $Parent(X_j)$

Algorithm for Nearly-Tree Structured CSPs

Conditioning: instantiate a variable, prune its neighbors' domains



Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree

- *Solve for each tree based on a specific value for cut variable (SA[r,b,g]*

Cutset size c $\Rightarrow$ runtime $O(d^c \cdot (n-c)d^2)$, very fast for small c

Summary

- CSP are a special class of search problems
  - States defined by a values for a fixed set of variables
  - Goal test defined by constraints on variables
  - Can be thought as a Depth-first search with one variable assigned per node
- Mini-Conflict heuristic based search is often a powerful tool for solving CSP problems
- Variable Ordering and Value Ordering heuristics can significantly speed-up search
- Constraint propagation can narrow the search space significantly
- If the problem can be mapped into a set of tree structured CSPs, it can be solved quickly

Next lecture

- Multi-level Search
  - Blackboard Problem-Solving Architecture