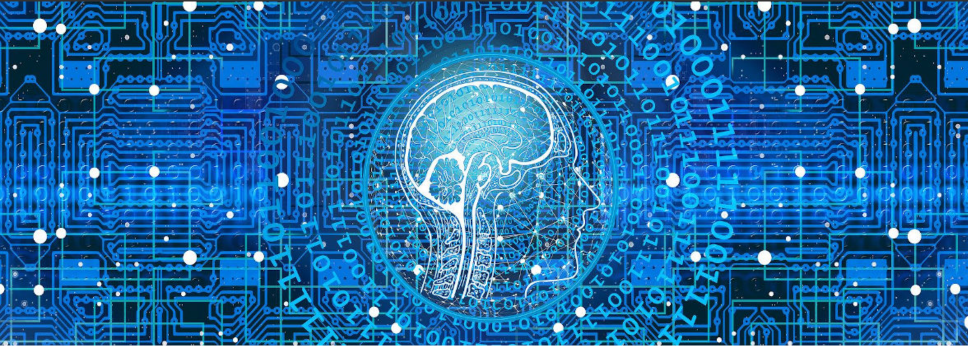


Chapter 5: Constraint Satisfaction Problems

Source A: Foundations of Artificial Intelligence by J. Boedecker, W. Burgard, F. Hutter, B. Nebel (Uni. Freiburg)

Contents

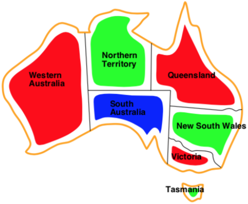
Constraint Satisfaction Problems

- A Constraint Satisfaction Problems (CSP) is given by
  - a set of variables $x_1, x_2, ..., x_n$,
  - an associated set of value domains $dom_1, dom_2, ..., dom_n$, and
  - a set of constraints. i.e., relations, over the variables.
- An assignment of values to variables that satisfies all constraints is a solution of such a CSP.
- In CSPs viewed as search problems, states are explicitly represented as variable assignments. CSP search algorithms take advantage of this structure.
- The main idea is to exploit the constraints to eliminate large portions of search space.
- Formal representation language with associated general inference algorithms

Example: Map-Coloring

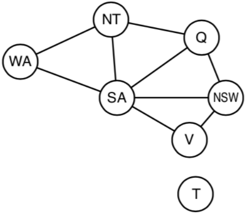
- Variables: WA,NT,SA,Q,NSW,V,T
- Values: red,green,blue
- Constraints: adjacent regions must have different colors, e.g., $NSW \neq V$

One Solution



- Solution assignment:
  - WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green

Constraint Graph



- a constraint graph can be used to visualize binary constraints
- for higher order constraints, hyper-graph representations might be used
- Nodes = variables, arcs = constraints

Note

Our problem is three-colorability for a planar graph

Variations

- Binary, ternary, or even higher arity (e.g., ALL DIFFERENT)
- Finite domains (d values) $\rightarrow d^n$ possible variable assignments
- Infinite domains (reals, integers)
  - linear constraints (each variable occurs only in linear form): solvable (in P if real)
  - nonlinear constraints: unsolvable

Applications

- Timetabling (classes, rooms, times)
- Configuration (hardware, cars, . . .)
- Spreadsheets
- Scheduling
- Floor planning
- Frequency assignments
- Sudoku
- ...

Backtracking Search over Assignments

- Assign values to variables step by step (order does not matter)
- Consider only one variable per search node!
- DFS (Deep First Search) with single-variable assignments is called *backtracking search*
- Can solve n-queens for $n \approx 25$

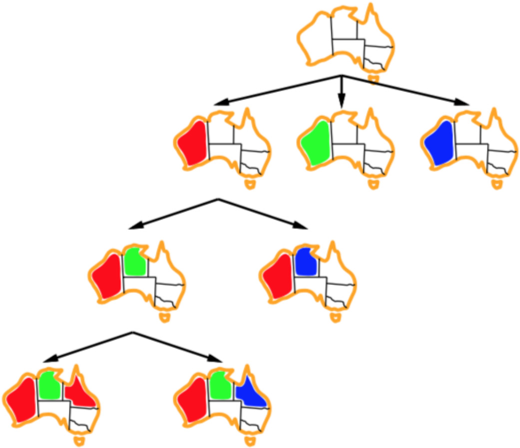
Algorithm

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
        return BACKTRACK({empty variable assignment}, csp)
```

```
function BACKTRACK(assignment,csp) returns a solution, or failure
        if assignment is complete then return assignment
        var ← SELECT-UNASSIGNED-VARIABLE(csp)
        for each value in ORDER-DOMAIN-VALUES(var,assignment,csp) do
                if value is consistent with assignment then
                        add var = value to assignment
                        inferences ← INFERENCE(csp , var , value )
                        if inferences≠failure then
                                add inferences to assignment
                                result←BACKTRACK(assignment,csp)
                                if result≠failure then
                                        return result
                remove var = value and inferences from assignment
        return failure
```

- ORDER-DOMAIN-VALUES : Im Prinzip geht jede Ordnung
- INFERENCE: Eine Funktion andere mögliche Zuordnungen ausschließt bzw. festlegt (später mehr)

Example



Improving Efficiency: CSP Heuristics & Pruning Techniques

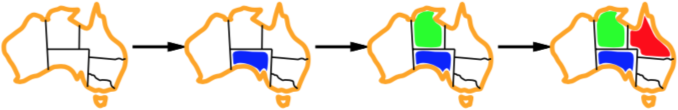
- Variable ordering: Which one to assign first?
- Value ordering: Which value to try first?
- Try to detect failures early on
- Try to exploit problem structure
- $\rightarrow$ Note: all this is not problem-specific!

Variable Ordering: Most constrained first

- Most constrained variable:
  - choose the variable with the fewest remaining legal values $\rightarrow$ reduces branching factor!

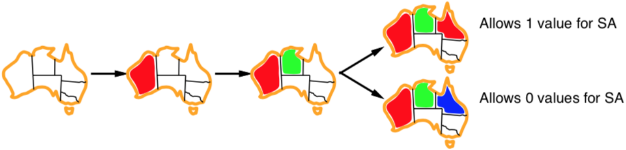
Variable Ordering: Most Constraining Variable First

- Break ties among variables with the same number of remaining legal values:
  - choose variable with the most constraints on remaining unassigned variables
  - $\rightarrow$ reduces branching factor in the next steps



Value Ordering: Least Constraining Value First

- Given a variable,
  - choose first a value that rules out the fewest values in the remaining unassigned variables
  - $\rightarrow$ We want to find an assignment that satisfies the constraints (of course, this does not help if the given problem is unsatisfiable.)



Rule out Failures early on: Forward Checking

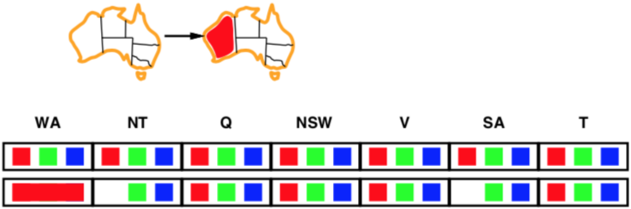
- Whenever a value is assigned to a variable, values that are now illegal for other variables are removed
- Implements what the ordering heuristics implicitly compute
- WA = red, then NT cannot become red
- If all values are removed for one variable, we can stop!

Forward Checking (1)

- Keep track of remaining values
- Stop if all have been removed

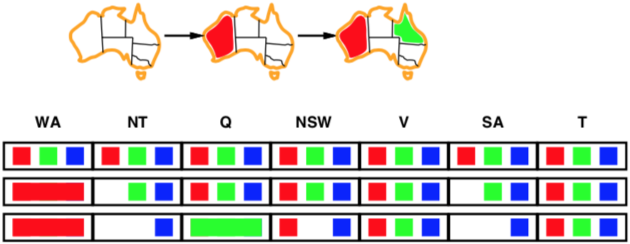
Forward Checking (2)

- Keep track of remaining values
- Stop if all have been removed



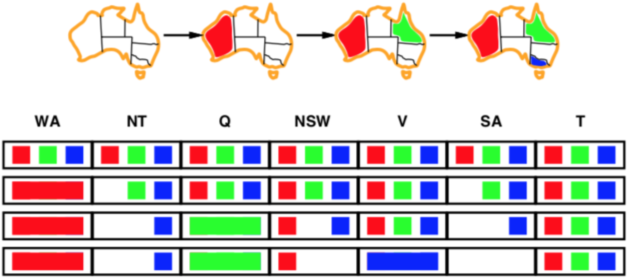
Forward Checking (3)

- Keep track of remaining values
- Stop if all have been removed



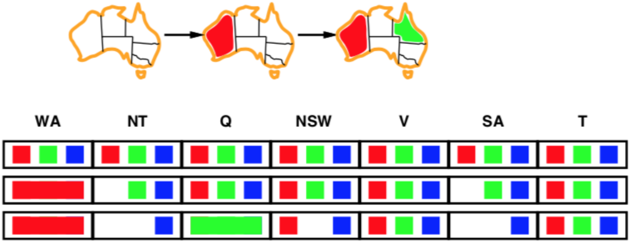
Forward Checking (4)

- Keep track of remaining values
- Stop if all have been removed



Forward Checking: Sometimes it Misses Something

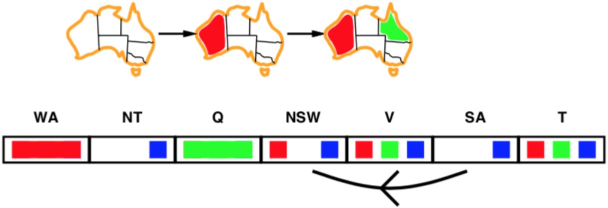
- Forward Checking propagates information from assigned to unassigned variables
- However, there is no propagation between unassigned variables



Arc Consistency

- A directed arc $X \rightarrow Y$ is "consistent" iff
  - for every value x of X, there exists a value y of Y, such that (x, y) satisfies the constraint between X and Y
- Remove values from the domain of X to enforce arc-consistency
- Arc consistency detects failures earlier
- Can be used as preprocessing technique or as a propagation step during backtracking

Arc Consistency Example



- enforcing Arc-Consistency between SA and NSW implies to remove blue from NSW
- enforcing Arc-Consistency between SA and NT, leaves no possible variable assignment in NSW.

AC-3 Algorithm

One example of an algorithm that enforces Arc-Consistency

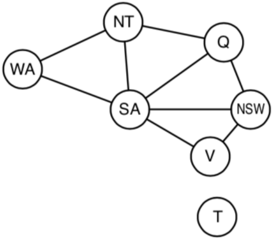
```
function AC-3(csp) returns false if an inconsistency is found and true otherwise
        inputs:  csp, a binary CSP with components (X, D, C)
        local variables:  queue, a queue of arcs, initially all the arcs in csp
        while queue is not empty do
                (X_i, X_j)←REMOVE-FIRST(queue)
                if REVISE(csp, X_i, X_j ) then
                        if size of D_i = 0 then return false
                        for each X_k in X_i.NEIGHBORS − {X_j} do
                                add(X_k, X_i) to queue
        return true
```

```
function REVISE(csp, X_i, X_j) returns true iff we revise the domain of X_i
        revised ← false
        for each x in D_i do
                if no value y in D_j allows (x,y) to sat-
                -isfy the constraint between X_i and X_j then
                        delete x from D_i
                        revised ← true
                return revised
```

Properties of AC-3

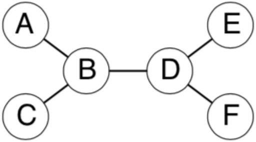
- AC-3 runs in O(d^3n^2) time, with n being the number of nodes and d being the maximal number of elements in a domain
  - good! only polynomial!
- Of course, AC-3 does not detect all inconsistencies (which is an NP-hard problem)

Problem Structure (1)



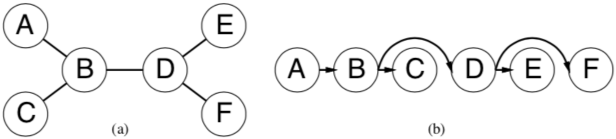
- CSP has two independent components
- Identifiable as connected components of constraint graph
- Can reduce the search space dramatically
  - One example here: T can be solved alone

Problem Structure (2): Tree-structured CSPs



- If the CSP graph is a tree, then it can be solved in $O(nd^2)$ (general CSPs need in the worst case $O(d^n)$).
- Idea: Pick root, order nodes, apply arc consistency from leaves to root, and assign values starting at root.

Problem Structure (2): Tree-structured CSPs

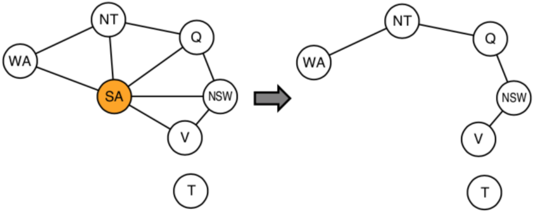


(a) (b)

- Pick any variable as root; choose an ordering such that each variable appears after its parent in the tree.
- Apply arc-consistency to (x_i, x_k) when x_i is the parent of x_k for all $k = n$ down to 2 (any tree with n nodes has $n - 1$ arcs and per arc d^2 comparisons are needed, which results in a complexity of $O(nd^2)$).
  - We start from the bottom (i.e F): First arc-consistency between F-D, i.e. remove values at D, then move upwards... at the end we know, that all possible assignments which are left, lead to a solution
- Now we can start at x_1 assigning values from the remaining domains without creating any conflict in one sweep through the tree!
- This algorithm is linear in n.

Problem Structure (3): Almost Tree-structured

- Idea: Reduce the graph structure to a tree by fixing values in a reasonably chosen subset

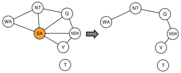


- Instantiate a variable and prune values in neighboring variables is called "Conditioning"

Problem Structure (4): Almost Tree-structured

Algorithm Cutset Conditioning:

- Choose a subset S of the CSPs variables such that the constraint graph becomes a tree after removal of S. The set S is called a **cycle cutset**.
- For each possible assignment of variables in S that satisfies all constraints on S
  - remove from the domains of the remaining variables any values that are inconsistent with the assignments for S, and
  - if the remaining CSP has a solution, return it together with the assignment for S

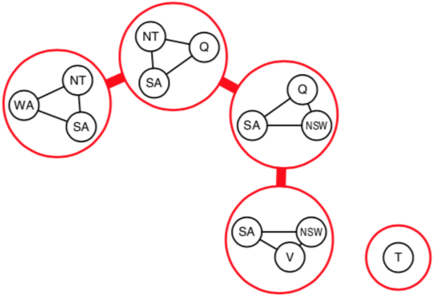


Note

Finding the smallest cycle cutset is NP hard, but several efficient approximation algorithms are known.

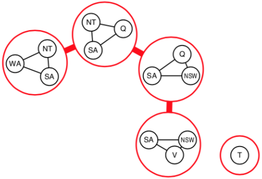
Another Method: Tree Decomposition (1/3)

- Decompose the problem into a set of connected sub-problems, where two sub-problems are connected when they share a constraint
- Solve the sub-problems independently and then combine the solutions



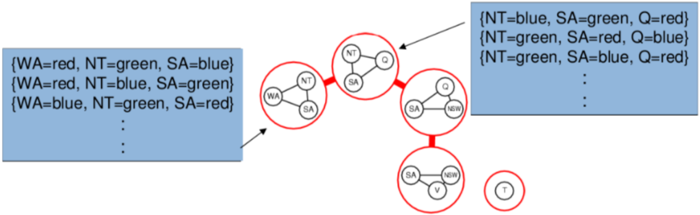
Another Method: Tree Decomposition (2/3)

- A tree decomposition must satisfy the following conditions:
  - Every variable of the original problem appears in at least one sub-problem
  - Every constraint appears in at least one sub-problem
  - If a variable appears in two sub-problems, it must appear in all sub-problems on the path between the two sub-problems
  - The connections form a tree



Another Method: Tree Decomposition (3/3)

- Consider sub-problems as new mega-variables, which have values defined by the solutions to the sub-problems
- Use technique for tree-structured CSP to find an overall solution (constraint is to have identical values for the same variable)



Tree Width

- The aim is to make the subproblems as small as possible. The tree width w of a tree decomposition is the size of largest sub-problem minus 1
- Tree width of a graph is minimal tree width over all possible tree decompositions
- If a graph has tree width w and we know a tree decomposition with that width, we can solve the problem in $O(nd^{w+1})$
- Unfortunately, finding a tree decomposition with minimal tree width is NP-hard. However, there are heuristic methods that work well in practice.

Example: Bavarian Map-Coloring Problem (1/2)

- Constraints: Make Bavarians happy
- So-Called AI-Hard Problem: solution requires true intelligence

Example: Bavarian Map-Coloring Problem (2/2)

- One possible starting point (might be already sufficient)

Summary and Outlook

- CSPs are a special kind of search problem:
  - states are value assignments
  - goal test is defined by constraints
- Backtracking = DFS with one variable assigned per node. Other intelligent backtracking techniques possible
- Variable/value ordering heuristics can help dramatically Constraint propagation prunes the search space
- Path-consistency is a constraint propagation technique for triples of variables
- Tree structure of CSP graph simplifies problem significantly
- Cutset conditioning and tree decomposition are two ways to transform part of the problem into a tree
- CSPs can also be solved using local search