Constraint Satisfaction Problems

Bùi Tiến Lên

01/09/2019

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Contents

1. Defining Constraint Satisfaction Problems
2. Backtracking Search for CSPs
3. Constraint Propagation: Inference in CSPs
4. Local Search for CSPs
5. The Structure of Problems

Defining Constraint Satisfaction Problems

Constraint Satisfaction Problems

- In standard search problem, **state** is a "black box"
- A **constraint satisfaction problem** (CSP) use a **factored representation** for each state.
  - State = a set of variables and each of which has a value
  - Solution = each variable has a value that satisfies all constraints on that variable
- Take advantage of the structure of states
- General-purpose rather than problem-specific heuristics
  - Identify combinations of variable-value that violate the constraints → eliminate large portions of the search space all at once
  - Solutions to complex problems

Constraint Satisfaction Problems (cont.)

A constraint satisfaction problem consists of three components, $\mathcal{X}$, $\mathcal{D}$, and $\mathcal{C}$:

- $\mathcal{X}$ is a set of variables, $\{X_1, \dots, X_n\}$
- $\mathcal{D}$ is a set of domains, $\{D_1, \dots, D_n\}$
  - Each domain D_i consists of a set of allowable values, $\{v_1, \dots, v_k\}$ for variable X_i
- $\mathcal{C}$ is a set of constraints that specify allowable combinations of values.
  - Each constraint C_j consists of a pair $\langle scope, rel \rangle$, where *scope* is a tuple of variables that participate in the constraint and *rel* is a relation that defines the values that those variables can take on

Defining Constraint Satisfaction Problems
Backtracking Search for CSPs
Constraint Propagation: Inference in CSPs
Local Search for CSPs
The Structure of Problems

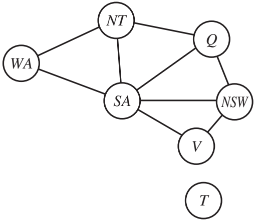
State Space And Solution

- Each state in a CSP 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 assigned
- A **partial assignment** is one that assigns values to only some of the variables.
- A **solution** to a CSP is a consistent, complete assignment.

Example problem: Map coloring

- The principal states and territories of Australia. Coloring this map can be viewed as a constraint satisfaction problem (CSP). The goal is to assign colors to each region so that no neighboring regions have the same color.

- The map-coloring problem represented as a constraint graph.
- **Constraint graph**: nodes are variables, arcs show constraints
- CSP algorithms use the graph structure to speed up search



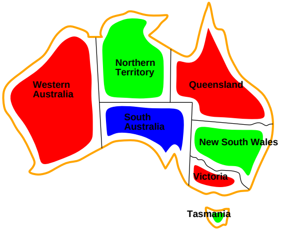
Example problem: Map coloring (cont.)

- **Variables**: $\mathcal{X} = \{WA, NT, Q, NSW, V, SA, T\}$
- **Domains**: $D_i = \{red, green, blue\}$
- **Constraints**: *adjacent regions must have different colors*

$$\mathcal{C} = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, \\ WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}$$

There are many possible solutions to this problem, e.g.,

$$\{WA = red, NT = green, Q = red, NSW = green, \\ V = red, SA = blue, T = green\}$$



Example problem: Job-shop scheduling

- We consider a small part of the car assembly, consisting of 15 tasks:
  - install axles (front and back)
  - affix all four wheels (right and left, front and back)
  - tighten nuts for each wheel
  - affix hubcaps
  - inspect the final assembly.
- Some tasks must occur before another while many other tasks can go on at once.
  - E.g., a wheel must be installed before the hubcap is put on
- A task takes a certain amount of time to complete.

Example problem: Job-shop scheduling (cont.)

We can represent the tasks with 15 variables:

- **Variables**:

$$\begin{aligned}\mathcal{X} = \{ & Axle_F, Axle_B, \\ & Wheel_{RF}, Wheel_{LF}, Wheel_{RB}, Wheel_{LB}, \\ & Nuts_{RF}, Nuts_{LF}, Nuts_{RB}, Nuts_{LB}, \\ & Cap_{RF}, Cap_{LF}, Cap_{RB}, Cap_{LB}, \\ & Inspect\}\end{aligned}$$

- **Domains**: D_i: the value of each variable is the time that the task starts

Example problem: Job-shop scheduling (cont.)

- **Constraints**: Assume task T_1 and T_2 take duration d_1 and d_2 to complete
  - **Precedence constraints**: task T_1 must occur before task T_2;

$$T_1 + d_1 \leq T_2$$

  - **Disjunctive constraint**: task T_1 and task T_2 must not overlap in time

$$T_1 + d_1 \leq T_2 \text{ or } T_2 + d_2 \leq T_1$$

Example problem: Job-shop scheduling (cont.)

- The axles have to be in place before the wheels are put on, and it takes 10 minutes to install an axle

$$\begin{array}{ll} Axle_F + 10 \leq Wheel_{RF} & Axle_F + 10 \leq Wheel_{LF} \\ Axle_B + 10 \leq Wheel_{RB} & Axle_B + 10 \leq Wheel_{LB} \end{array}$$

- For each wheel, we must affix the wheel (which takes 1 minute), then tighten the nuts (2 minutes), and finally attach the hubcap (1 minute)

$$\ldots$$

- Suppose we have four workers to install wheels, but they have to share one tool that helps put the axle in place

$$\ldots$$

Example problem: Job-shop scheduling (cont.)

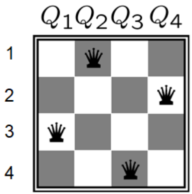
- The inspection comes last and takes 3 minutes

$$\forall X \neq Inspect, X + d_X \leq Inspect$$

- Finally, suppose there is a requirement to get the whole assembly done in 30 minutes $\rightarrow$ limit the domain of all variables to

$$D_j = \{1, 2, 3, \ldots, 27\}$$

Example problem: 4-Queens

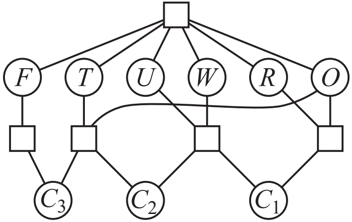


- **Variables**: $\mathcal{X} = \{Q_1, Q_2, Q_3, Q_4\}$
- **Domains**: $D_i = \{1, 2, 3, 4\}$
- **Constraints**:
  $Q_i \neq Q_j$ (cannot be in same row)
  $|Q_i - Q_j| \neq |i - j|$ (or same diagonal)

Example problem: Cryptarithmetic

$$\begin{array}{cccc} & T & W & O \\ + & T & W & O \\ \hline F & O & U & R \end{array}$$

(a)



(b)

- **Variables**: $\mathcal{X} = \{F, T, U, W, R, O, C_1, C_2, C_3\}$
- **Domains**: $D_i = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- **Constraints**:
  - $Alldiff(F, T, U, W, R, O)$
  - $T \neq 0, F \neq 0$
  - $C_3 = F, \ldots$

Why formulate a problem as a CSP

- Provide natural representation for a wide variety of problems
- Many problems **intractable** in regular state-space search can be solved quickly with CSP formulation.
  For example, once we have chosen $\{SA = blue\}$ in the Australia problem.
  - Search: $3^5 = 243$ assignments
  - CSP: $2^5 = 32$ assignments
- Better insights to the problem and its solution

Varieties of CSPs

- Discrete variables and finite domains
  - n variables with size $d \to O(n^d)$ complete assignments
  - E.g., Boolean CSPs, including Boolean satisfiability (NP-complete)
- Discrete variables and infinite domains
  - E.g., job scheduling, variables are start/end times for each job
- Continuous variables
  - E.g., start/end times for Hubble Telescope observations

Varieties of constraints

- **Unary** constraints involve a single variable, e.g., $SA \neq green$
- **Binary** constraints involve pairs of variables, e.g., $SA \neq WA$
- **Higher-order** constraints involve 3 or more variables, e.g., cryptarithmetic column constraints
- **Preferences** (soft constraints), e.g., *red* is better than *green* often representable by a cost for each variable assignment $\rightarrow$ **constrained optimization problems**

Real-world CSPs

- Assignment problems; e.g., who teaches what class?
- Timetabling problems; e.g., which class is offered when and where?
- Hardware configuration
- Spreadsheets
- Transportation scheduling
- Factory scheduling
- Floorplanning

Backtracking Search for CSPs

Defining Constraint Satisfaction Problems
Backtracking Search for CSPs
Constraint Propagation: Inference in CSPs
Local Search for CSPs
The Structure of Problems

CSP as Standard Search Problem

Let's start with the straightforward, dumb approach, then fix it

- States are defined by the values assigned so far
  - **Initial state**: the empty assignment, $\emptyset$
  - **Successor function**: assign a value to an unassigned variable that does not conflict with current assignment $\implies$ fail if no legal assignments (not fixable!)
  - **Goal test**: the current assignment is complete

1. This is the same for all CSPs!
2. Every solution appears at depth n with n variables $\implies$ use depth-first search
3. Path is irrelevant, so can also use complete-state formulation
4. Branching factor $b = (n - \ell)d$ at depth ℓ, hence $n!d^n$ leaves!

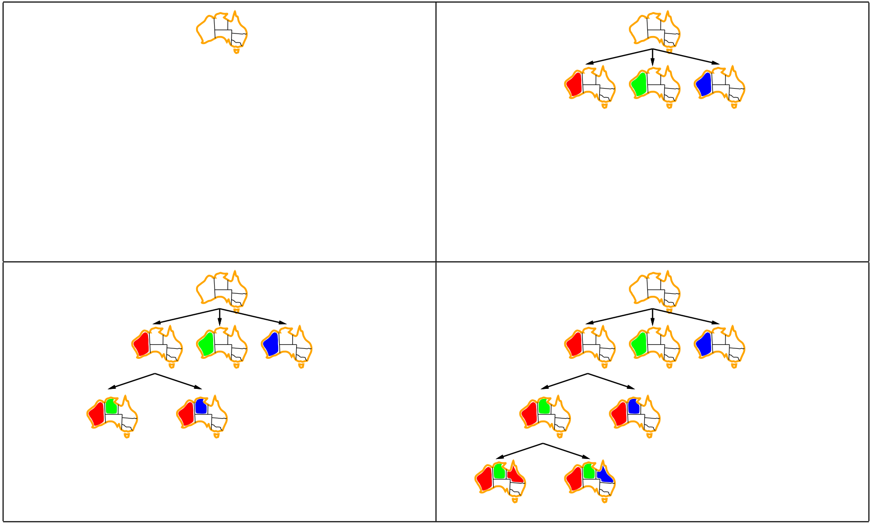
Backtracking search

- Variable assignments are **commutative**, i.e., [$WA = red$ then $NT = green$] same as [$NT = green$ then $WA = red$]
- Only need to consider assignments to a single variable at each node $\implies$ $b = d$ and there are d^n leaves
- Depth-first search for CSPs with single-variable assignments is called **backtracking** search
- Backtracking search is the basic uninformed algorithm for CSPs
- Can solve n-queens for $n \approx 25$

Algorithm

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
 return BACKTRACK(∅, 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
```

Illustration



Improving backtracking efficiency

General-purpose methods can give huge gains in speed:

1. Which variable should be assigned next?
2. In what order should its values be tried?
3. Can we detect inevitable failure early?
4. Can we take advantage of problem structure?

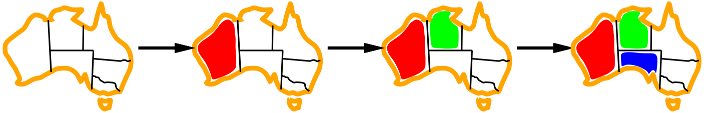
Minimum remaining values

The backtracking algorithm contains the line

$$\mathit{var} \leftarrow \textsc{Select-Unassigned-Variable}(\mathit{csp})$$

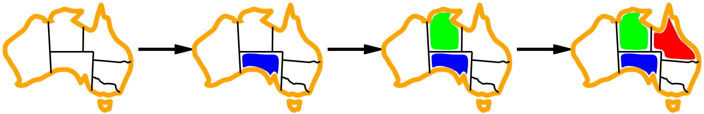
We need a strategy for $\textsc{Select-Unassigned-Variable}$ to choose the next unassigned variable in $\{X_1, X_2, \ldots\}$

- **Minimum remaining values** (MRV): choose the variable with the fewest legal values
- MRV usually performs better than a random/static ordering, sometimes by a factor of 1,000 or more



Degree heuristic

- Tie-breaker among MRV variables
- **Degree heuristic**: choose the variable with the most constraints on remaining variables

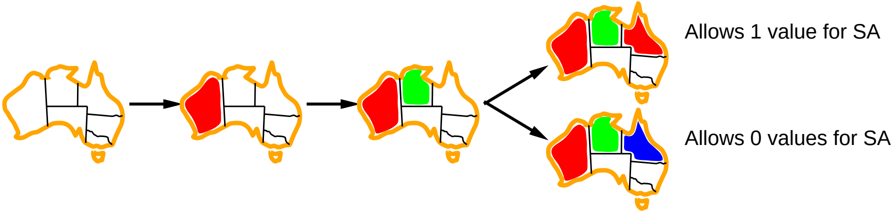


Least constraining value

The backtracking algorithm contains the line

$$\textit{value} \text{ in } \textsc{Order-Domain-Values}(\textit{var}, \textit{assignment}, \textit{csp})$$

- Given a variable, choose the least constraining value: the one that rules out the fewest values in the remaining variables



- Combining these heuristics makes 1000 queens feasible

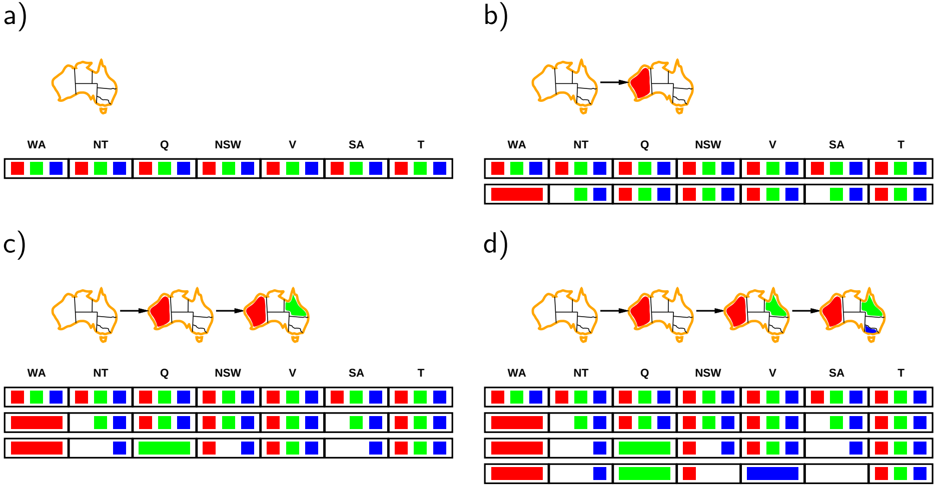
Inference: Forward checking

- **Idea**: Keep track of remaining legal values for unassigned variables
  $\rightarrow$ Terminate search when any variable has no legal values

$$\mathit{inferences} \leftarrow \textsc{Inference}(\mathit{csp}, \mathit{var}, \mathit{value})$$

- Note: Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures!

Inference: Forward checking



Constraint Propagation: Inference in CSPs

Defining Constraint Satisfaction Problems

Backtracking Search for CSPs

Constraint Propagation: Inference in CSPs

Local Search for CSPs

The Structure of Problems

Constraint propagation

- Reduce the number of legal values for a variable by using constraints $\rightarrow$ legal values for another variable also reduced
- Intertwined with search, or done as a preprocessing step
  - Sometimes the preprocessing can solve the whole problem!
- Enforcing local consistency in each part of a graph causes inconsistent values to be eliminated throughout the graph

Node consistency

- A single variable is **node-consistent** if all the values in the variable's domain satisfy the variable's **unary constraints**
  - South Australians dislike green,
    $SA = \{red, green, blue\} \rightarrow SA = \{red, blue\}$
- Eliminate all the unary constraints in a CSP by running node consistency

Arc consistency

- A variable in a CSP is arc-consistent if every value in its domain satisfies the variable's **binary constraints**. More formally,

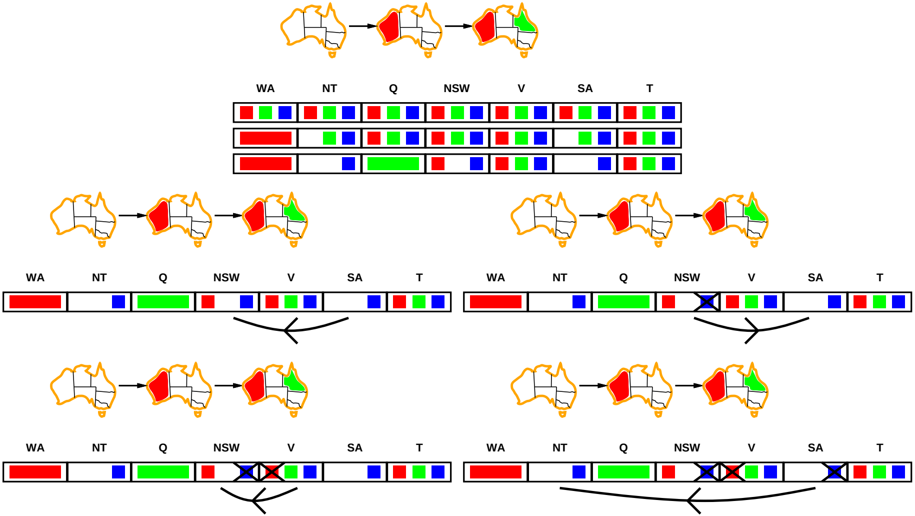
  $X \to Y$ is consistent iff for *every* value x of X there is *some* allowed y
- Run as a preprocessor before the search starts or after each assignment, must be run repeatedly until no inconsistency remains.
- Trade-off
  - Requires some overhead to do, but generally more effective than direct search
  - Eliminate large (inconsistent) parts of the state-space more effectively than search
- Need a systematic method for arc-checking
  - If Y loses a value, neighbors of Y need to be rechecked $\to$ incoming arcs can become inconsistent again while outgoing arcs stay still

Algorithm

```
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 ≠ ∅ 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 satisfy
       the constraint between X_i and X_j then
      delete x from D_i
      revised ← true
  return revised
```

Illustration



Local Search for CSPs

Defining Constraint Satisfaction Problems

Backtracking Search for CSPs

Constraint Propagation: Inference in CSPs

Local Search for CSPs

The Structure of Problems

Local search for CSPs

- Complete-state formulation
  - The initial state assigns a value to every variable $\rightarrow$ violation
  - The search changes the value of one variable at a time $\rightarrow$ eliminate the violated constraints
- Min-conflicts heuristic: the minimum number of conflicts with other variables
- Min-conflicts is surprisingly effective for many CSPs.
  - Million-queens problem can be solved $\sim$ 50 steps
  - Hubble Space Telescope: the time taken to schedule a week of observations down from 3 weeks to $\sim$ 10 minutes

Algorithm

```
function MIN-CONFLICTS(csp, max_steps) returns a solution of failure
inputs: csp, a constraint satisfaction problem
        max_steps, the number of steps allowed before giving up
  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 csp.VARIABLES
    value ← the value v for var that minimizes CONFLICTS(var, v, current, csp)
    set {var = value} in current
  return failure
```

Performance of min-conflicts

- Given random initial state, can solve *n*-queens in almost constant time for arbitrary *n* with high probability (e.g., $n = 10,000,000$)
- The same appears to be true for any randomly-generated CSP *except* in a narrow range of the ratio

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$

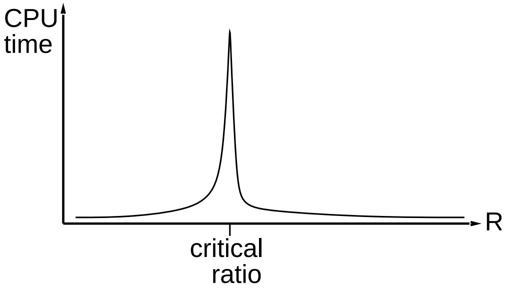


Illustration: 8-queens

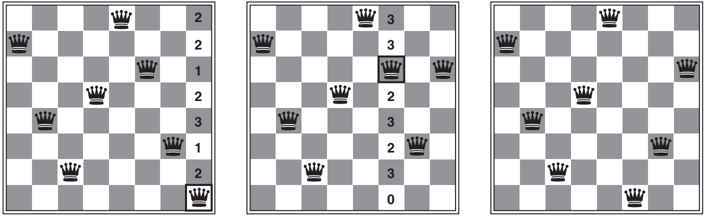


Figure 1: A two-step solution using min-conflicts for an 8-queens problem. At each stage, a queen is chosen for reassignment in its column. The number of conflicts (in this case, the number of attacking queens) is shown in each square. The algorithm moves the queen to the min-conflicts square, breaking ties randomly.

Improving min-conflicts

- The landscape of a CSP under the min-conflicts heuristic usually has a series of plateaux.
  - Millions of variable assignments that are only one conflict away from a solution
- **Plateau search**: allow sideways moves to another state with the same score
- **Tabu search**: keep a small list of recently visited states and forbid the algorithm to return to those states
- **Simulated annealing** can also be used

Constraint weighting

- Concentrate the search on the important constraints
- Each constraint is given a numeric weight, W_i, initially all 1.
- At each step, chooses a variable/value pair to change that has the lowest total weight of all violated constraints.
- Increase the weight of each constraint that is violated by the current assignment.

Local search in online setting

- Scheduling problems: online setting
  - A week's airline schedule may involve thousands of flights and tens of thousands of personnel assignments
  - The bad weather at one airport can render the schedule infeasible.
- The schedule should be repaired with a minimum number of changes.
  - Done easily with a local search starting from the current schedule
  - A backtracking search with the new set of constraints usually requires much more time and might find a solution with many changes from the current schedule

The Structure of Problems

Independent subproblems

- A problem CSP is decomposed into **independent subproblems** CSP_i
  - Independent subproblems are identifiable as connected components of constraint graph

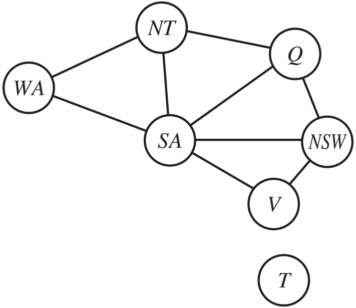


Figure 2: Tasmania and mainland are **independent subproblems**

Independent subproblems (cont.)

- If assignmen S_i is a solution of CSP_i then

$$S = \bigcup_{i=1} S_i \text{ is a solution of } CSP$$

- Suppose a problem of n variables can be broken into independent subproblems of only c variables
  - Worst-case solution cost is $O(\frac{n}{c} d^c)$
  - Original CSP has worst-case solution $O(d^n)$

Defining Constraint Satisfaction Problems

Backtracking Search for CSPs

Constraint Propagation: Inference in CSPs

Local Search for CSPs

The Structure of Problems

Tree-structured CSPs

- A CSP is a tree-structured CSP if the constraint graph has no loops

Theorem 1

A tree-structured CSP can be solved in $O(n \times d^2)$ time, compare to general CSPs, where worst-case time is $O(d^n)$

Proof

self-exercise ∎

Tree-structured CSPs (cont.)

- Using TOPOLOGICALSORT to create an ordering of the variables such that each variable appears after its parent in the tree

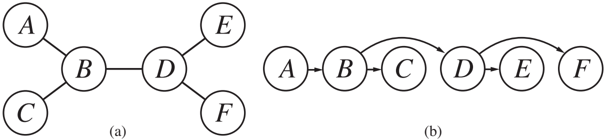


Figure 3: (a) The constraint graph of a tree-structured CSP. (b) A linear ordering of the variables consistent with the tree with A as the root. This is known as a **topological sort** of the variables.

Algorithm

```
function TREE-CSP-SOLVER(csp) returns a solution, or failure
inputs: csp, a CSP with components 𝒳, 𝒟, 𝒞
  n ← |𝒳|
  assignment ← ∅
  root ← any variable in 𝒳
  𝒳 ← TOPOLOGICALSORT(𝒳, root)
  for j = n down to 2 do
    MAKE-ARC-CONSISTENT(PARENT(X_j), X_j)
    if it cannot be made consistent then return failure
  for i = 1 to n do
    assignment[X_i] ← any consistent value from D_i
    if there is no consistent value then return failure
  return assignment
```

Nearly tree-structured CSPs

- **Cycle cutset**: a set of variables such that the remaining constraint graph is a tree

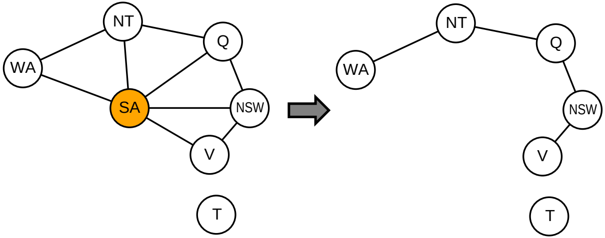


Figure 4: The removal of *SA* makes the constraint graph to be a tree

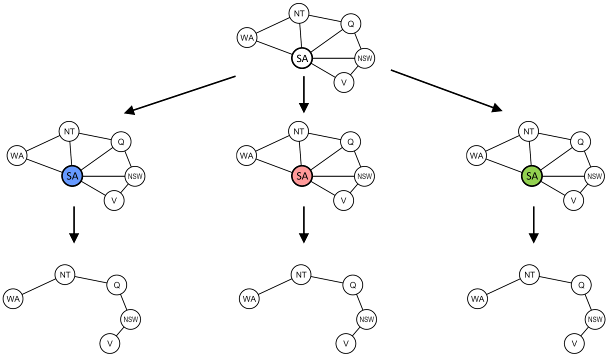
Cutset Conditioning

Choose a cutset

Instantiate the cutset (all possible ways)

Compute residual CSP for each assignment

Solve the residual CSPs (tree structured)



- If the cycle cutset has size c, then the total run time is $O(d^c(n-c)d^2)$

Defining Constraint Satisfaction Problems

Backtracking Search for CSPs

Constraint Propagation: Inference in CSPs

Local Search for CSPs

The Structure of Problems

Tree Decomposition

- **Idea**: create a tree-structured graph of mega-variables
  - Each mega-variable encodes part of the original CSP
  - Subproblems overlap to ensure consistent solutions
- A **tree decomposition** must satisfy the following three requirements:
  1. Every variable in the original problem appears in at least one of the subproblems.
  2. 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 subproblems.
  3. If a variable appears in two subproblems in the tree, it must appear in every subproblem along the path connecting those subproblems.

Tree Decomposition (cont.)

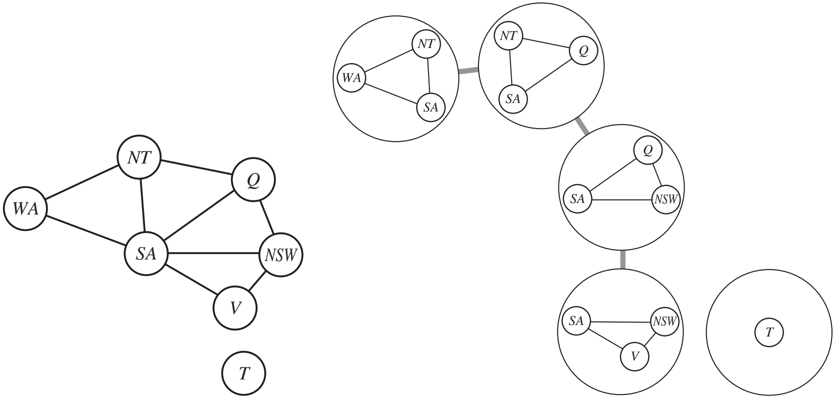


Figure 5: A tree decomposition of the constraint graph

The structure of values

- Consider the map-coloring problem with n colors.
- For every consistent solution, there is actually a set of $n!$ solutions formed by permuting the color names.
  - E.g., WA, NT, and SA must all have different colors, but there are 3! ways to assign the three colors to these three regions.
- **Symmetry-breaking constraint**: Impose an arbitrary ordering constraint that requires the values to be in alphabetical order.
  - E.g., $NT < SA < WA$ $\rightarrow$only one solution possible $\{NT = blue, SA = green, WA = red\}$

References

Goodfellow, I., Bengio, Y., and Courville, A. (2016).
Deep learning.
MIT press.

Nguyen, T. (2018).
Artificial intelligence slides.
Technical report, HCMC University of Sciences.

Russell, S. and Norvig, P. (2016).
Artificial intelligence: a modern approach.
Pearson Education Limited.