Constraint Satisfaction Problem (CSP)

II

★ K-Consistency

⇒ Increasing degree of Consistency!

* 1-Consistency (Node Consistency)

⇒ Every Single node's domain has a value which meets the node's Unary Constraints.

* 2-Consistency (Arc Consistency)

⇒ For each pair of nodes, any Consistent assignment to one can be extended to the other.

* K-Consistency

⇒ For each k nodes, any Consistent assignment to K-1 can be extended to the k^{th} node.

★ Strong K-Consistency

↳ Also K-1, K-2 ---- Consistent.

⇒ Claim: Strong n-Consistency means we can Solve without backtracking!

3-Consistency ⇒ Path Consistency

★ Structure

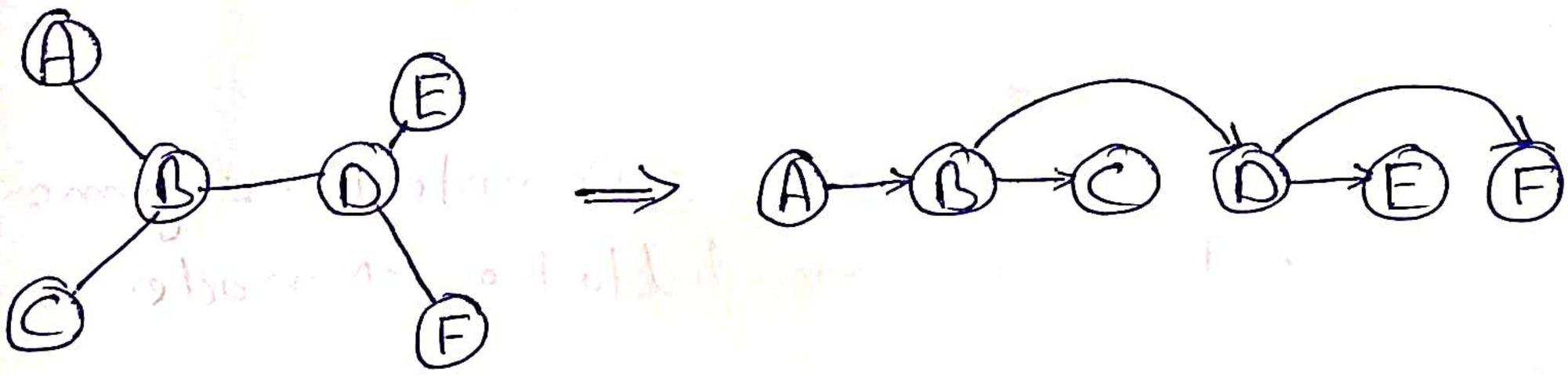
⇒ Suppose a graph of n variables can be broken into subproblems of only C variables.

★ Tree-Structured CSP

Theorem: If the constraint graph has no loops, the CSP can be solved in $O(nd^2)$ time.

⇒ Algorithm for tree-structured CSPs:

① Order: Choose a root variable, order variables so that parents precede children.

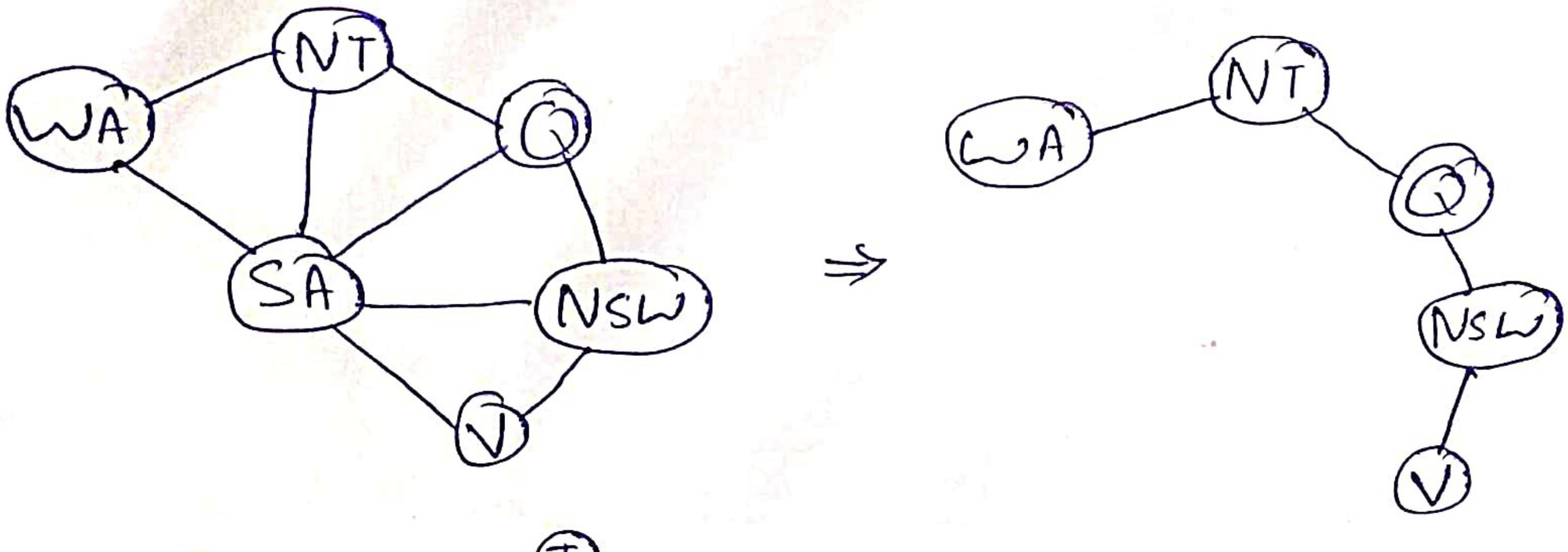


② Remove backward: For $i = n:2$, apply RemoveInConsistent(Parent(X_i), X_i)

③ Assign forward: For $i = 1:n$ assign X_i consistently with Parent(X_i)

Runtime: $O(nd^2)$

★ Nearly Tree-Structured CSPs



(T)

(T)

⇒ 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.

⇒ Cutset size c gives runtime $O(d^c(n-c)d^2)$.

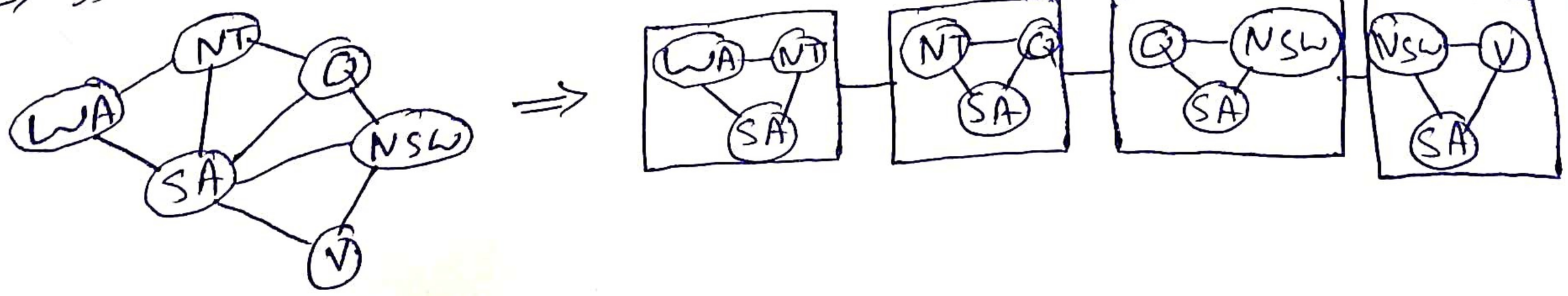
↳ Very fast for small C.

★ Tree Decomposition*

⇒ Idea: Create a tree-structured graph of mega-variables

⇒ Each mega-variables encodes part of the original CSP.

⇒ Sub problem overlap to ensure consistent solutions.



* Iterative Improvement

⇒ Local Search methods typically work with "Complete state" i.e. all variables assigned.

⇒ To apply to CSPs:

- → Take an assignment with Unsatisfied Constraints
- → Operators reassign variable values
- → No fringe! Live on the edge.

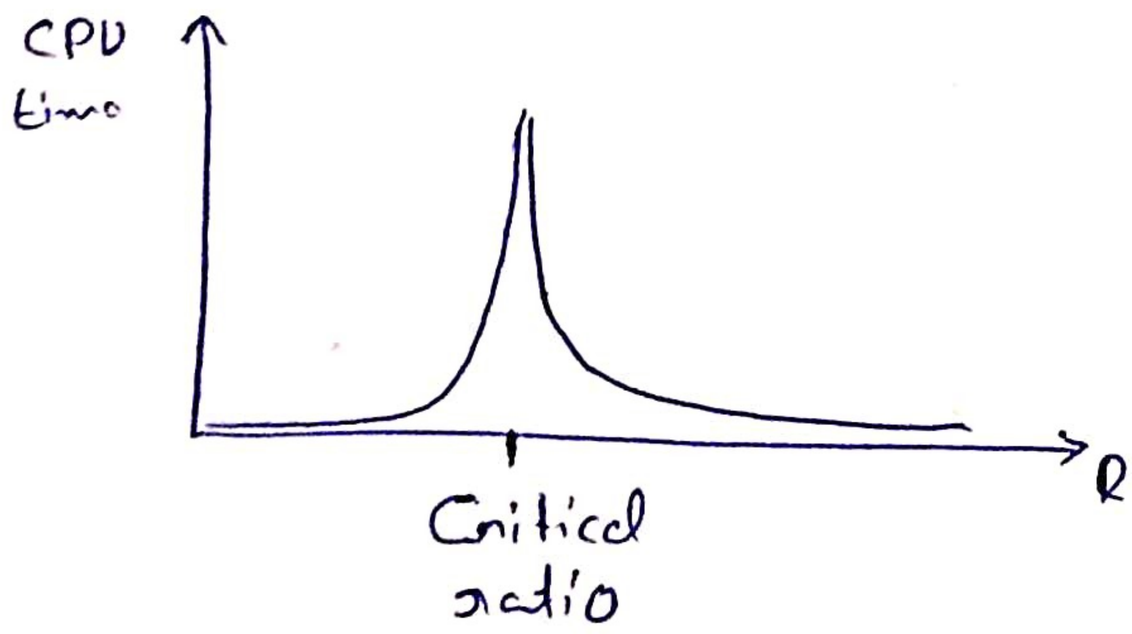
⇒ Algorithm: While not Solved,

→ Variable Selection: Randomly Select any Conflicted variable

→ Value Selection: Min-Conflicts heuristic

- → Choose a value that violates the fewest Constraints
- → Hill climb with $h(n)$ = Total number of Violated Constraints.

$$R = \frac{\text{Number of Constraints}}{\text{number of Variables}}$$



* Local Search

⇒ Tree search keeps unexplored alternatives on the fringe (ensures completeness)

⇒ Local search: Improve a single option until you can't make it better (no fringe)

↳ Generally much faster & more memory efficient (but incomplete & suboptimal)

(Simulated Annealing)

{" Escape local maximum by allowing downhill moves "}

(Genetic Algorithms)