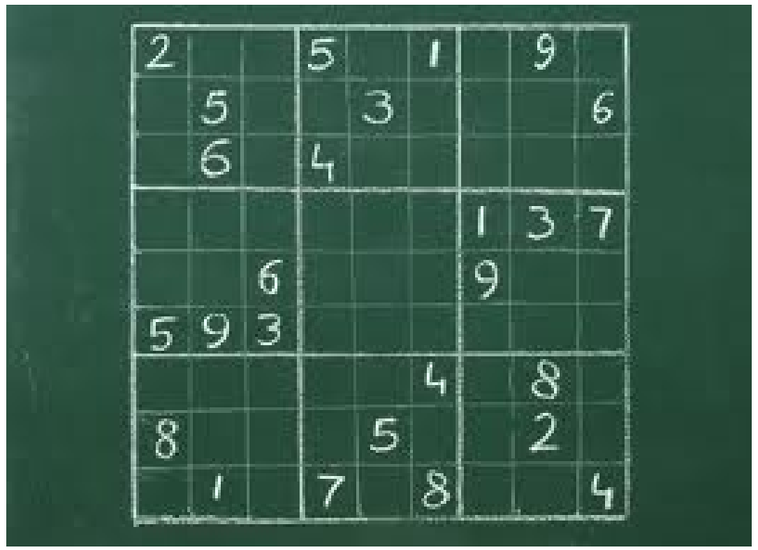
CSPs: arc consistency



- In this module, we will introduce the notion of arc consistency, which will lead us to a lookahead algorithm called AC-3 for pruning domains and speeding up backtracking search.

Review: backtracking search

Algorithm: backtracking search

Backtrack$(x, w, \text{Domains})$:

- If x is complete assignment: update best and return
- Choose unassigned **VARIABLE** X_i (MCV)
- Order **VALUES** Domain_i of chosen X_i (LCV)
- For each value v in that order:
  - $\delta \leftarrow \prod_{f_j \in D(x, X_i)} f_j(x \cup \{X_i : v\})$
  - If $\delta = 0$: continue
  - $\text{Domains}' \leftarrow$ Domains via **LOOKAHEAD** (AC-3)
  - If any $\text{Domains}'_i$ is empty: continue
  - Backtrack$(x \cup \{X_i : v\}, w\delta, \text{Domains}')$

- So far, we have seen the backtracking search algorithm for finding the maximum weight assignment in a CSP.
- Given a partial assignment, we find an unassigned variable, assign a value, and then perform lookahead, which tries to prune down the domains as much as possible.
- We saw forward checking, a lookahead algorithm where we prune the domains of the neighbors of X_i. We will now upgrade this to AC-3 (forward checking without brakes).
- Good lookahead serves two important purposes:
- First, smaller domains lead to smaller branching factors, which makes search faster. In the exteme case, a domain of size 0 represents an inconsistency in the partial assignment, allowing us to backtrack.
- A second motivation is that since the domain sizes are used in the context of the dynamic ordering heuristics (most constrained variable and least constrained value), we can hope to choose better orderings with domains that more accurately reflect what values are actually possible.

Arc consistency: example

Example: numbers

Before enforcing arc consistency on X_i:

$X_i \in \text{Domain}_i = \{1, 2, 3, 4, 5\}$

$X_j \in \text{Domain}_j = \{1, 2\}$

Factor: $[X_i + X_j = 4]$

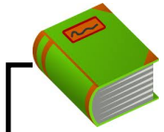
After enforcing arc consistency on X_i:

$X_i \in \text{Domain}_i = \{2, 3\}$

X_i	1	2	3	4	5
X_j	1	2			

- To build up to AC-3, we need to introduce the idea of arc consistency.
- To enforce arc consistency on X_i with respect to X_j, we go through each of the values in the domain of X_i and remove it if there is no value in the domain of X_j that is consistent with X_i.
- For example, $X_i = 4$ is ruled out because no value $X_j \in \{1, 2\}$ satisfies $X_i + X_j = 4$.

Arc consistency



Definition: arc consistency

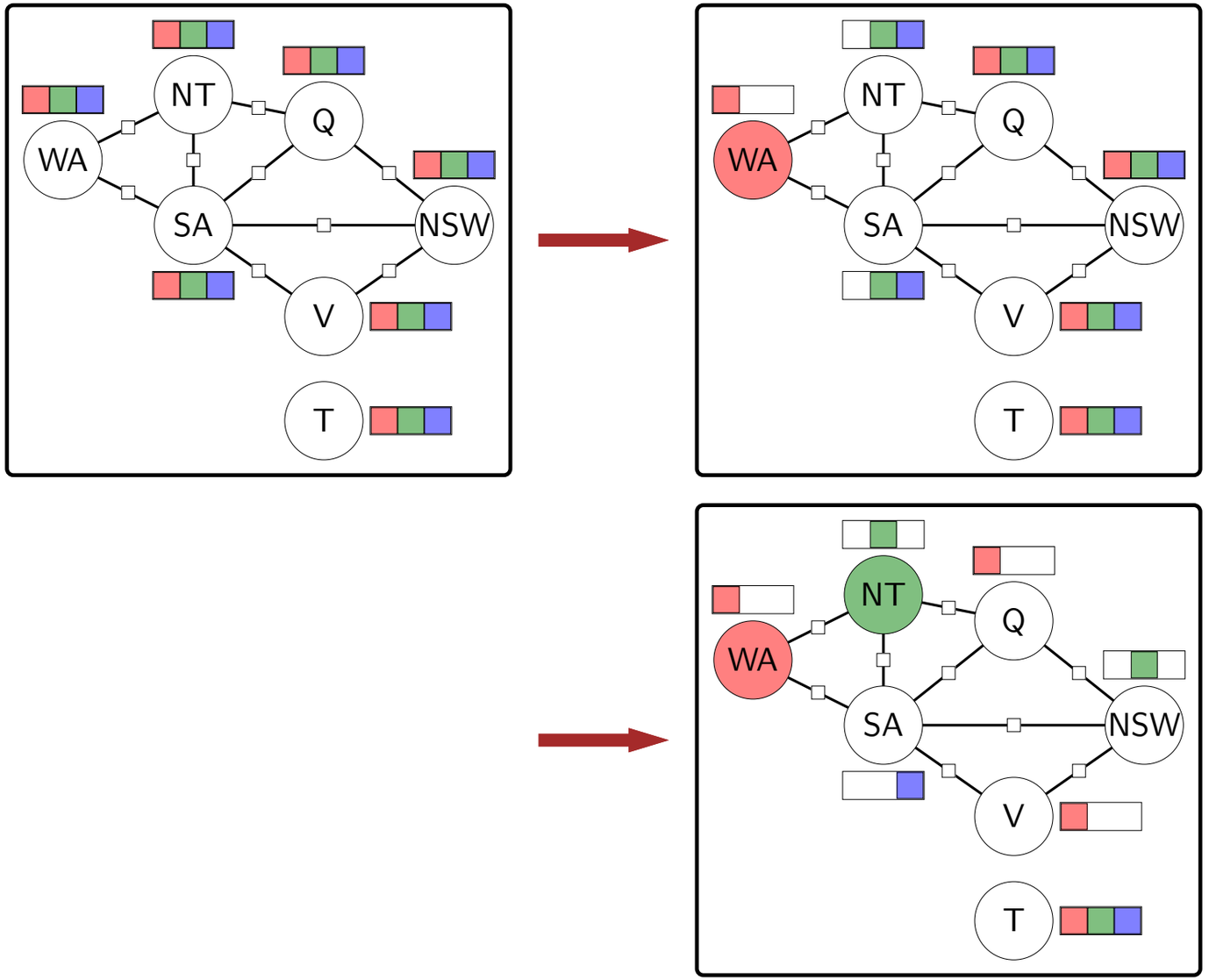
A variable X_i is **arc consistent** with respect to X_j if for each $x_i \in \text{Domain}_i$, there exists $x_j \in \text{Domain}_j$ such that $f(\{X_i : x_i, X_j : x_j\}) \neq 0$ for all factors f whose scope contains X_i and X_j.

Algorithm: enforce arc consistency

EnforceArcConsistency(X_i, X_j): Remove values from Domain_i to make X_i arc consistent with respect to X_j.

- Formally, a variable X_i is arc consistent with respect to X_j if every value in the domain of X_i has some potential partner in the domain of X_j.
- Enforcing arc consistency just makes it so.

AC-3 (example)



- The AC-3 algorithm simply enforces arc consistency until no domains change. Let's walk through this example.
- We start with the empty assignment.
- Suppose we assign WA to R. Then we enforce arc consistency on the neighbors of WA. Those domains change, so we enforce arc consistency on their neighbors, but nothing changes. Note that at this point AC-3 produces the same output as forward checking.
- We recurse and suppose we now pick NT and assign it G. Then we enforce arc consistency on the neighbors of NT (which is forward checking), SA, Q, NSW.
- AC-3 converges at this point, but note how much progress we've made: we have have nailed down the assignment for all the variables (except T)!
- Importantly, though we've touched all the variables, we are still at the NT node in the search tree. We've just paved the way, so that when we recurse, there's only one value to try for SA, Q, NSW, and V!

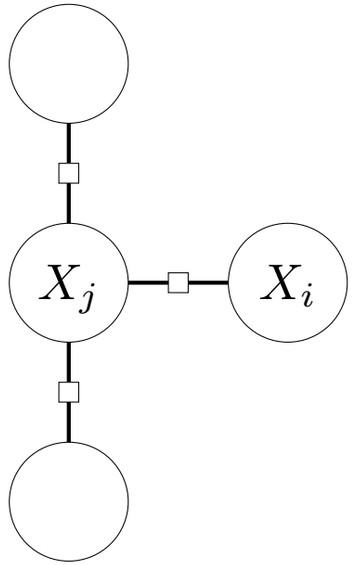
AC-3

Forward checking: when assign $X_j : x_j$, set $\text{Domain}_j = \{x_j\}$ and enforce arc consistency on all neighbors X_i with respect to X_j

AC-3: repeatedly enforce arc consistency on all variables

Algorithm: AC-3

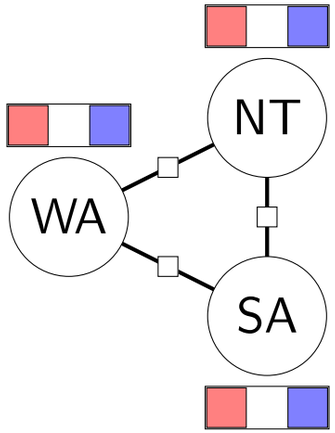
$S \leftarrow \{X_j\}$.
While S is non-empty:
    Remove any X_j from S.
    For all neighbors X_i of X_j:
        Enforce arc consistency on X_i w.r.t. X_j.
        If Domain_i changed, add X_i to S.



- In forward checking, when we assign a variable X_i to a value, we are actually enforcing arc consistency on the neighbors of X_i with respect to X_i.
- Why stop there? AC-3 doesn't. In AC-3, we start by enforcing arc consistency on the neighbors of X_i (forward checking). But then, if the domains of any neighbor X_j changes, then we enforce arc consistency on the neighbors of X_j, etc.
- Note that unlike BFS graph search, a variable could get added to the set multiple times because its domain can get updated more than once. More specifically, we might enforce arc consistency on (X_i, X_j) up to D times in the worst case, where $D = \max_{1 \leq i \leq n} |\text{Domain}_i|$ is the size of the largest domain. There are at most m different pairs (X_i, X_j) and each call to enforce arc consistency takes $O(D^2)$ time. Therefore, the running time of this algorithm is $O(ED^3)$ in the very worst case where E is the number of edges (though usually, it's much better than this).

Limitations of AC-3

- AC-3 isn't always effective:



- No consistent assignments, but AC-3 doesn't detect a problem!

- Intuition: if we look locally at the graph, nothing blatantly wrong...

- The previous example showed that AC-3 essentially solves the CSP, but this is too good to be true in general.
- Here is an example of a simplified factor graph, where running AC-3 won't prune any of the domains. While the domains are not empty, there is no solution.
- AC-3 really only spot checks the domains locally. Locally, everything looks fine, even though there's no global solution (impossible to assign two colors to 3 provinces).
- Advanced: We could generalize arc consistency to fix this problem. Instead of looking at every 2 variables and the factors between them, we could look at every subset of k variables, and check that there's a way to consistently assign values to all k, taking into account all the factors involving those k variables. However, there is a substantial cost to doing this (the running time is exponential in k in the worst case), so generally arc consistency ($k = 2$) is good enough.

Summary

- Enforcing arc consistency: make domains consistent with factors
- Forward checking: enforces arc consistency on neighbors
- AC-3: enforces arc consistency on neighbors and their neighbors, etc.
- Lookahead very important for backtracking search!

- In summary, we presented the idea of enforcing arc consistency, which prunes domains based on information from a neighboring variable.
- After assigning a variable, forward checking enforces arc consistency on its neighbors, while AC-3 does it to the neighbors of neighbors, etc.
- Recall that AC-3 (or any lookahead algorithm) is used in the context of backtracking search to reduce the branching factor and keeps the dynamic ordering heuristics accurate. It can make a big difference!