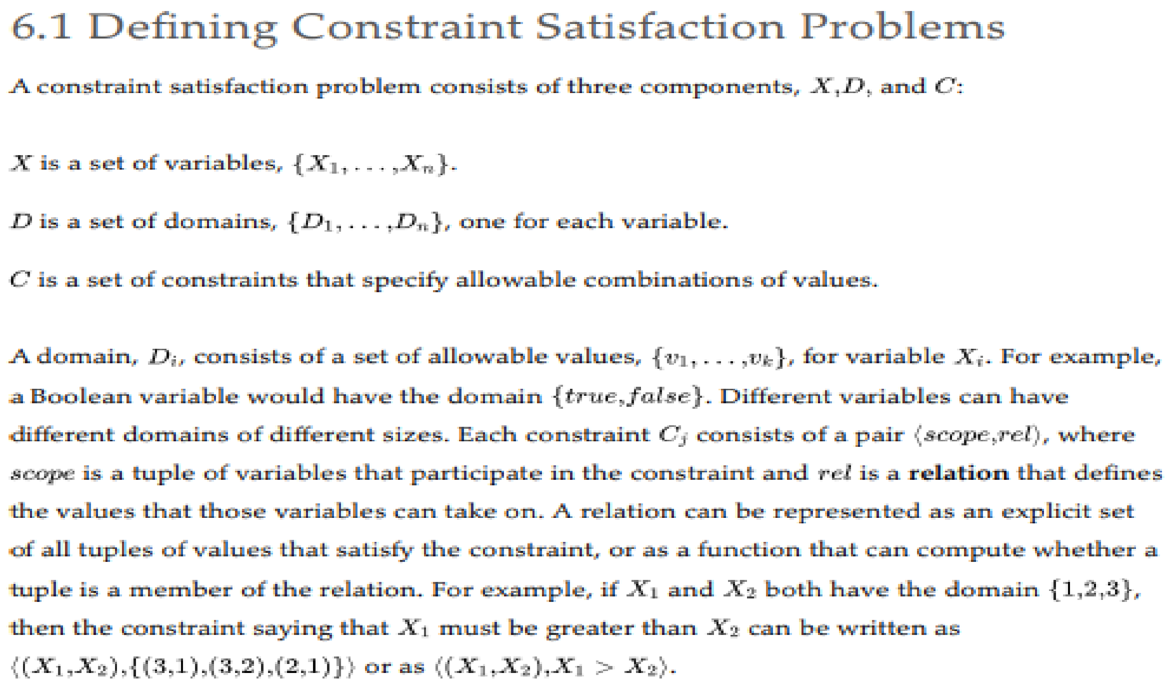
Chapter 6: Constraint Satisfaction

(Since we studied this material by going through the Moore tutorial, make sure you review those slides
first.)

6.1 Defining CSPs

• The Map coloring problem was covered directly in class, in some detail, as was job-shop. You should be able to formulate new problems that are variants of those.



6.1 Defining Constraint Satisfaction Problems

A constraint satisfaction problem consists of three components, X, D, and C:

X is a set of variables, $\{X_1, \ldots, X_n\}$.

D is a set of domains, $\{D_1, \ldots, D_n\}$, one for each variable.

C is a set of constraints that specify allowable combinations of values.

A domain, D_i, consists of a set of allowable values, $\{v_1, \ldots, v_k\}$, for variable X_i. For example, a Boolean variable would have the domain $\{true, false\}$. Different variables can have different domains of different sizes. 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. A relation can be represented as an explicit set of all tuples of values that satisfy the constraint, or as a function that can compute whether a tuple is a member of the relation. For example, if X_1 and X_2 both have the domain $\{1,2,3\}$, then the constraint saying that X_1 must be greater than X_2 can be written as $\langle (X_1, X_2), \{(3,1),(3,2),(2,1)\} \rangle$ or as $\langle (X_1, X_2), X_1 > X_2 \rangle$.

Unary constraint

In addition to examining the types of variables that can appear in CSPs, it is useful to look at the types of constraints. The simplest type is the unary constraint, which restricts the value of a single variable. For example, in the map-coloring problem it could be the case that South Australians won't tolerate the color green; we can express that with the unary constraint . (The initial specification of the domain of a variable can also be seen as a unary constraint.)

Binary constraint

A binary constraint relates two variables. For example, is a binary constraint. A binary CSP is one with only unary and binary constraints; it can be represented as a constraint graph.

Defining CSP:

CSP represent a state with a set of variable/ value pair and represent the condition for a solution by a set of constraints on the variables.

Why to formulate problem as a CSP?

1. Yield a natural representation for a wide variety of problems.
2. Solver can be faster than the state space searcher because the csp solver cam quickly eliminate large swatches of the search space.

UANRY CONSTARINT:

Restrict the value of a single variable

Eg. South Aus won’t tolerate red we can express that with unary constraint {SA =/ red}

BINARY CONSTRAINT

Relates two variables

Eg. {SA =/ NSW}

GLOBAL CONSTARINT

Involves an arbitrary number of variables

Eg. Alldiff constraint used in sudoku problems

6.2 Inference in CSP

• Node consistency, arc consistency, path consistency all covered

The process of using the constraints to reduce the number of legal values for a variables which in turn can reduce the legal values for another variable and so on.

The key idea is **LOCAL CONSISTENCY**:

1. NODE CONSTARINT:

A single variable is node consistent if all the values in the variables domain should satisfy the variable unary constraint.

A network is node consistent if every variable in the network is node consistent

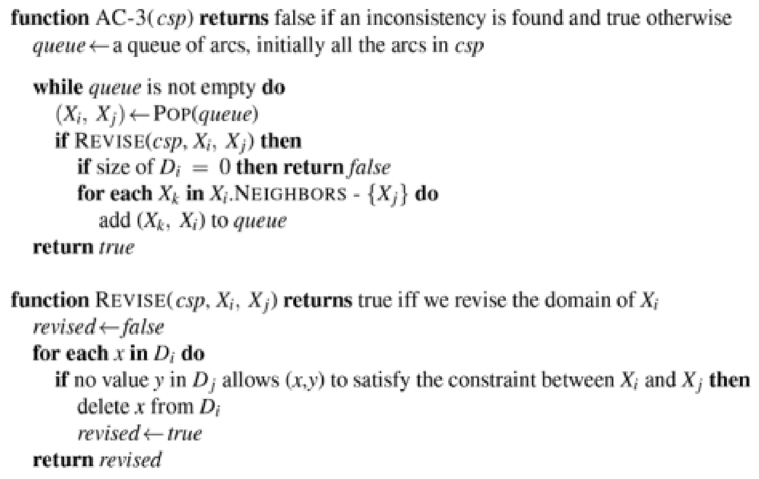
REDUECING THE DOMAIN ONE BY ONE

2. ARC CONSITENCY

A variable where two variables are involved.

Fulfill binary Constraints

Algo – AC-3



```
function AC-3(csp) returns false if an inconsistency is found and true otherwise
  queue ← a queue of arcs, initially all the arcs in csp

  while queue is not empty do
    $(X_i, X_j)$ ← POP(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
```

PATH CONSISTENCY

Arc consistency tightens down the domains (unary constraints) using the arcs (binary constraints). To make progress on problems like map coloring, we need a stronger notion of consistency. Path consistency tightens the binary constraints by using implicit constraints that are inferred by looking at triples of variables

• 6.2.4 K-consistency, skipped
• 6.2.5 and 6.2.6 – global constraints
Alldiff
6.3 Backtracking Search
• Backtracking search covered in the Moore tutorial

In backtracking search for Constraint Satisfaction Problems (CSPs), we need to decide which variable to assign next and in what order to try its possible values. Here's a simplified explanation of variable and value ordering:

1. Variable Ordering:

   - This is about deciding which variable to assign a value to next.

   - The goal is to pick the variable that is most likely to lead to a solution or detect a failure quickly.

   - Minimum-Remaining-Values (MRV) Heuristic: Choose the variable with the fewest legal values left.

     - Helps prune the search tree by selecting variables that are most constrained first.

     - Also known as the "fail-first" heuristic because it picks variables that are likely to cause a failure soon.

   - Degree Heuristic: Select the variable that is involved in the largest number of constraints on other unassigned variables.

     - Helps reduce the branching factor on future choices.

     - Useful as a tie-breaker when multiple variables have the same MRV.

   - Example: If a region in a map has only one color option left, it's best to assign a color to that region next.

2. Value Ordering:

   - Once a variable is chosen, we need to decide which value to assign to it.

   - The goal is to pick a value that leaves the maximum flexibility for subsequent variable assignments.

   - Least-Constraining-Value Heuristic: Choose the value that rules out the fewest choices for neighboring variables.

     - Helps maintain flexibility and avoid limiting options for neighboring variables.

   - Example: If assigning a certain color to a region eliminates options for its neighboring regions, it's better to choose a different color.

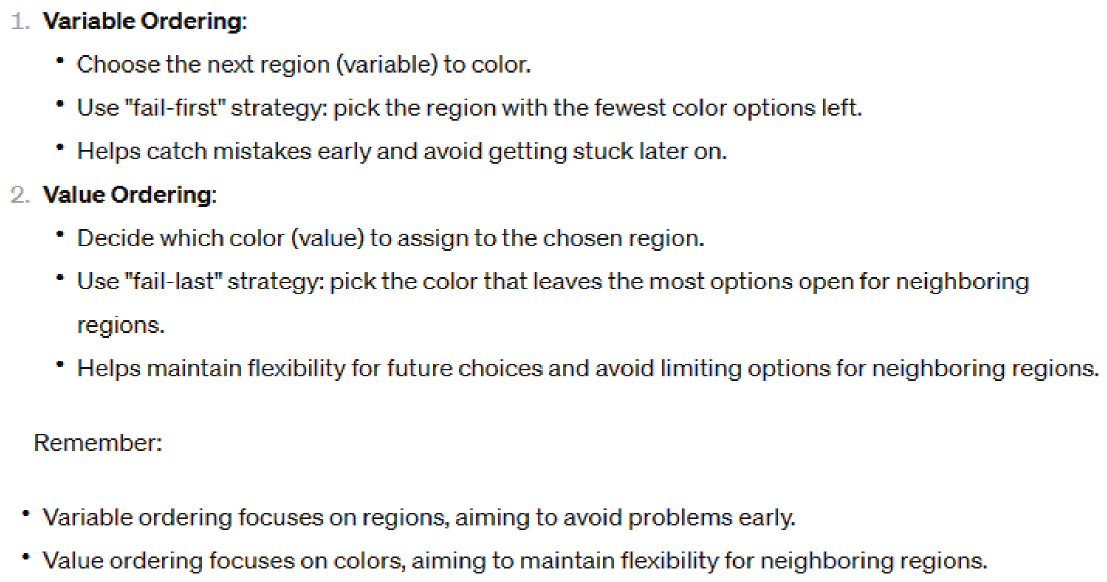
Remember:

- Variable ordering focuses on choosing the next variable to assign, aiming to detect failures early.

- Value ordering focuses on selecting the value for the chosen variable, aiming to maintain flexibility for neighboring variables.

By using these heuristics, we can guide the backtracking search more effectively, reducing the number of failed assignments and leading to faster solutions.

• 6.3.1 Variable and value ordering – concepts were covered, but we did not cover examples



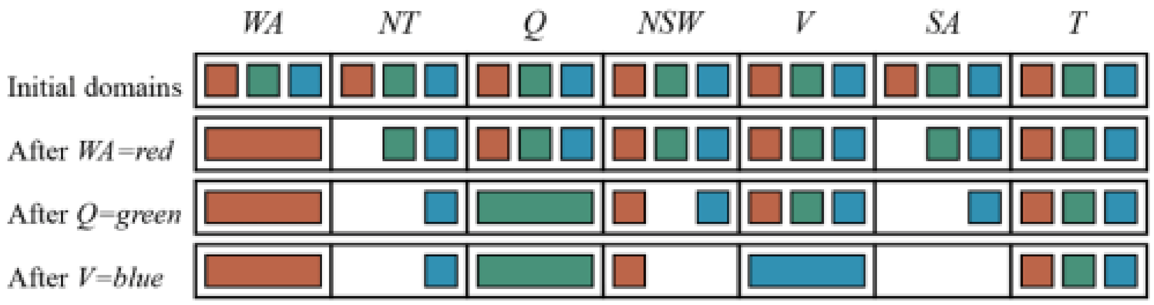
1. **Variable Ordering**:
   - Choose the next region (variable) to color.
   - Use "fail-first" strategy: pick the region with the fewest color options left.
   - Helps catch mistakes early and avoid getting stuck later on.
2. **Value Ordering**:
   - Decide which color (value) to assign to the chosen region.
   - Use "fail-last" strategy: pick the color that leaves the most options open for neighboring regions.
   - Helps maintain flexibility for future choices and avoid limiting options for neighboring regions.

Remember:

- Variable ordering focuses on regions, aiming to avoid problems early.
- Value ordering focuses on colors, aiming to maintain flexibility for neighboring regions.

• 6.3.2 Interleaving search and inference – forward checking – covered

Figure 6.7



The progress of a map-coloring search with forward checking. $WA = red$ is assigned first; then forward checking deletes red from the domains of the neighboring variables NT and SA. After $Q = green$ is assigned, $green$ is deleted from the domains of NT, SA, and NSW. After $V = blue$ is assigned, $blue$ is deleted from the domains of NSW and SA, leaving SA with no legal values.

- 6.3.3 Skip

6.4 and 6.5 Skip