**Constraint satisfaction problems**

A constraint satisfaction problem (CSP) is a problem that requires its solution within some limitations/conditions also known as constraints. It consists of the following:

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

* X is a set of variables, {X1, . . . ,Xn}.
* D is a set of domains, {D1, . . . ,Dn}, one for each variable.
* C is a set of constraints that specify allowable combinations of values.

Please note that the elements in the domain can be both continuous and discrete but in AI, we generally only deal with discrete values.

Also, note that all these sets should be finite except for the domain set. Each variable in the variable set can have different domains. For example, consider the Sudoku problem again. Suppose that a row, column and block already have 3, 5 and 7 filled in. Then the domain for all the variables in that row, column and block will be {1,2,4,6,8,9}.

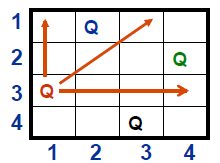
**Popular Problems of CSP**

The following problems are some of the popular problems that can be solved using CSP:

* CryptArithmetic (Coding alphabets to numbers.)
* n-Queen (In an n-queen problem, n queens should be placed in a nXn matrix such that no queen shares the same row, column or diagonal.)
* Map Coloring (Coloring different regions of map ensuring no adjacent regions have the same color.)
* Crossword (Everyday puzzles appearing in newspapers.)
* Sudoku (A number grid.)
* Latin Square Problem

**Eg.**

1. N-Queens: Place N queens on an NxN chessboard so that none can attack the other.



Variables: board positions in NxN chessboard

Domains: Queen or blank

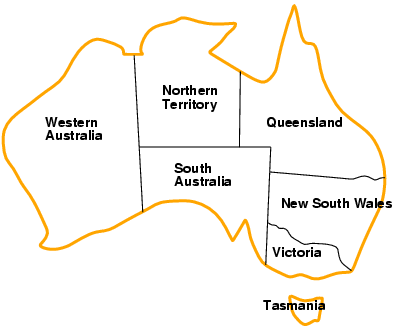
Constraints: Two positions on a line (vertical, horizontal, diagonal) cannot both be Q

1. Graph Coloring: Pick colors for map regions, avoiding coloring adjacent regions with the same color.

Variables: regions

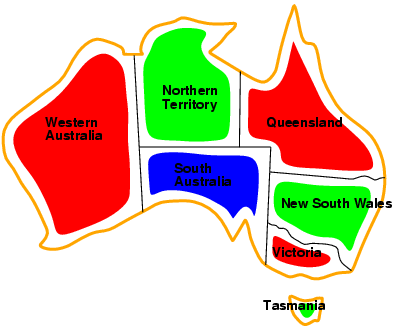
Domains: colors allowed

Constraints: adjacent regions must have different colors.

Eg.

* Variables *WA, NT, Q, NSW, V, SA, T*
* Domains *Di* = {red,green,blue}
* Constraints: adjacent regions

must have different colors  
e.g., WA ≠ NT, or (WA,NT) in {(red,green),(red,blue),(green,red), (green,blue),(blue,red),(blue,green)}



Solution:

Solutions are complete and consistent assignments.

for example, WA = red, NT = green, Q = red,

NSW = green,V = red,SA = blue,T = green

**Variations on the CSP formalism:**

* Discrete variables
  + finite domains:
    - *n* variables, domain size *d 🡪 O(dn)* complete assignments
    - e.g., Boolean CSPs, incl. Boolean satisfiability (NP-complete)
  + infinite domains:
    - integers, strings, etc.
    - e.g., job scheduling, variables are start/end days for each job
    - need a constraint language, e.g., *StartJob1 + 5 ≤ StartJob3*
* Continuous variables
  + e.g., start/end times for Hubble Space Telescope observations

linear constraints solvable in polynomial time by LP.

* Unary constraints involve a single variable,
  + e.g., SA ≠ green
* Binary constraints involve pairs of variables,
  + e.g., SA ≠ WA
  + Higher-order constraints involve 3 or more variables,
  + e.g., cryptarithmetic column constraints

**CONSTRAINT PROPAGATION:**

In regular state-space search, an algorithm can do only search. In CSPs there is a choice: an algorithm can search or do a specific type of inference called constraint propagation, using the constraints to reduce the number of legal values for a variable, which in turn can reduce the legal values for another variable, and so on. Constraint propagation may be intertwined with search, or it may be done as a pre-processing step, before search starts.

The key idea is local consistency. If we treat each variable as a node in a graph and each binary constraint as an arc, then the process of enforcing local consistency in each part of the graph causes inconsistent values to be eliminated throughout the graph. There are different types of local consistency.

**Node consistency:** A single variable corresponding to a node in the CSP network is node-consistent if all the values in the variable’s domain satisfy the variable’s unary constraints.

Eg. In the variant of the Australia map-coloring problem where South Australians dislike green, the variable SA starts with domain {red , green, blue}, and we can make it node consistent by eliminating green, leaving SA with the reduced domain {red , blue}.

We say that a network is node-consistent if every variable in the network is node-consistent. It is always possible to eliminate all the unary constraints in a CSP by running node consistency. It is also possible to transform all n-ary constraints into binary ones.

**Arc consistency:** A variable in a CSP is arc-consistent if every value in its domain satisfies the variable’s binary constraints. More formally, Xi is arc-consistent with respect to another variable Xj if for every value in the current domain Di there is some value in the domain Dj that satisfies the binary constraint on the arc (Xi, Xj). A network is arc-consistent if every variable is arc consistent with every other variable.

Eg.

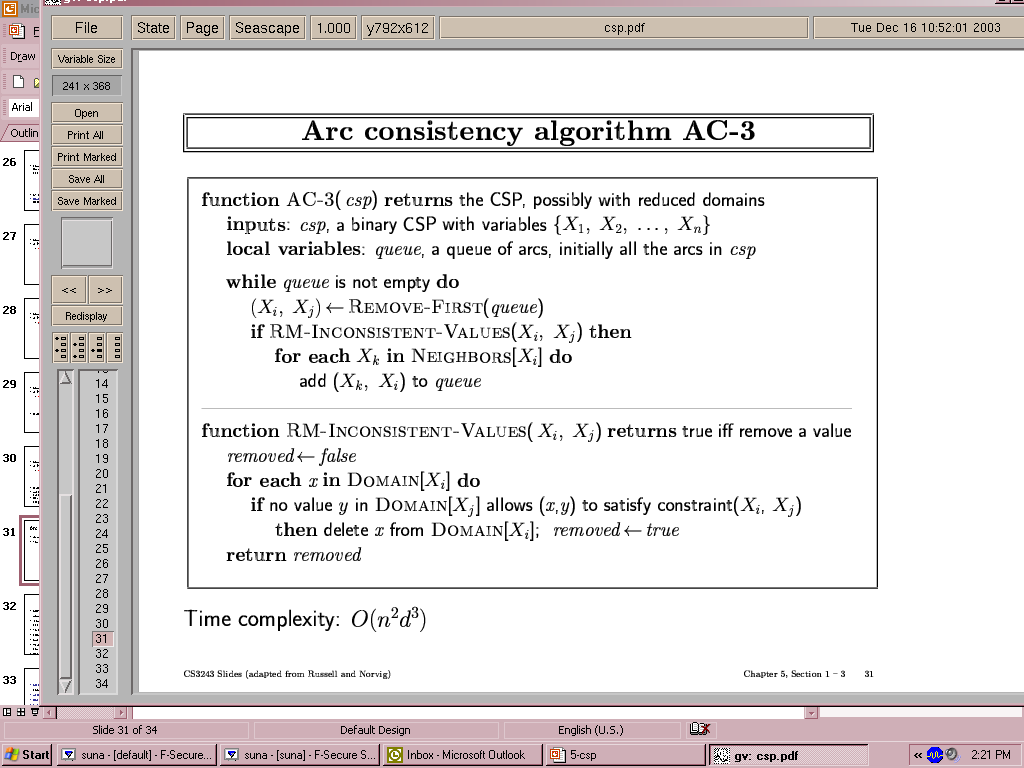
1. consider the constraint Y = X2 where the domain of both X and Y is the set of digits. We can write this constraint explicitly as (X, Y ), {(0, 0), (1, 1), (2, 4), (3, 9))}.

To make X arc-consistent with respect to Y, we reduce X’s domain to {0, 1, 2, 3}. If we also make Y arc-consistent with respect to X, then Y’s domain becomes {0, 1, 4, 9} and the whole CSP is arc-consistent.

1. Arc consistency can do nothing for the Australia map-coloring problem.

Consider the following inequality constraint on (SA,WA):

{(red , green), (red , blue), (green, red ), (green, blue), (blue, red ), (blue, green)}. No matter what value you choose for SA (or for WA), there is a valid value for the other variable. So applying arc consistency has no effect on the domains of either variable. The most popular algorithm for arc consistency is called AC-3.



* Time complexity: O(#constraints|domain|3)

Drawback:

Arc consistency can go a long way toward reducing the domains of variables, sometimes finding a solution by reducing every domain to size 1 and sometimes finding that the CSP

cannot be solved by reducing some domain to size 0.

**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.

A two-variable set {Xi, Xj} is path-consistent with respect to a third variable Xm if, for every assignment {Xi = a, Xj = b} consistent with the constraints on {Xi, Xj}, there is an assignment to Xm that satisfies the constraints on {Xi, Xm} and {Xm, Xj}. This is called path consistency because one can think of it as looking at a path from Xi to Xj with Xm in the middle.

*K***-consistency:**

* A CSP is *k-consistent* if, for any set of k-1 variables, and for any consistent assignment to those variables, a consistent value can always be assigned to any kth variable
* 1-consistency is node consistency
* 2-consistency is arc consistency
* For binary constraint networks, 3-consistency is the same as *path consistency*
* Getting k-consistency requires time and space exponential in k
* *Strong k-consistency* means k-consistency for all k from 1 to k
  + Once strong k-consistency for k=#variables has been obtained, solution can be constructed trivially
* Tradeoff between propagation and branching
* Practitioners usually use 2-consistency and less commonly 3-consistency