University of Malta
BSc(Hons.) IT Year II

Problem Solving Techniques in Artificial Intelligence

Lecture 13.1

Search

- Constraint Satisfaction Problems
- Node and Arc Consistency

Department of Computer Science and AI
1998/9

Constraint Satisfaction

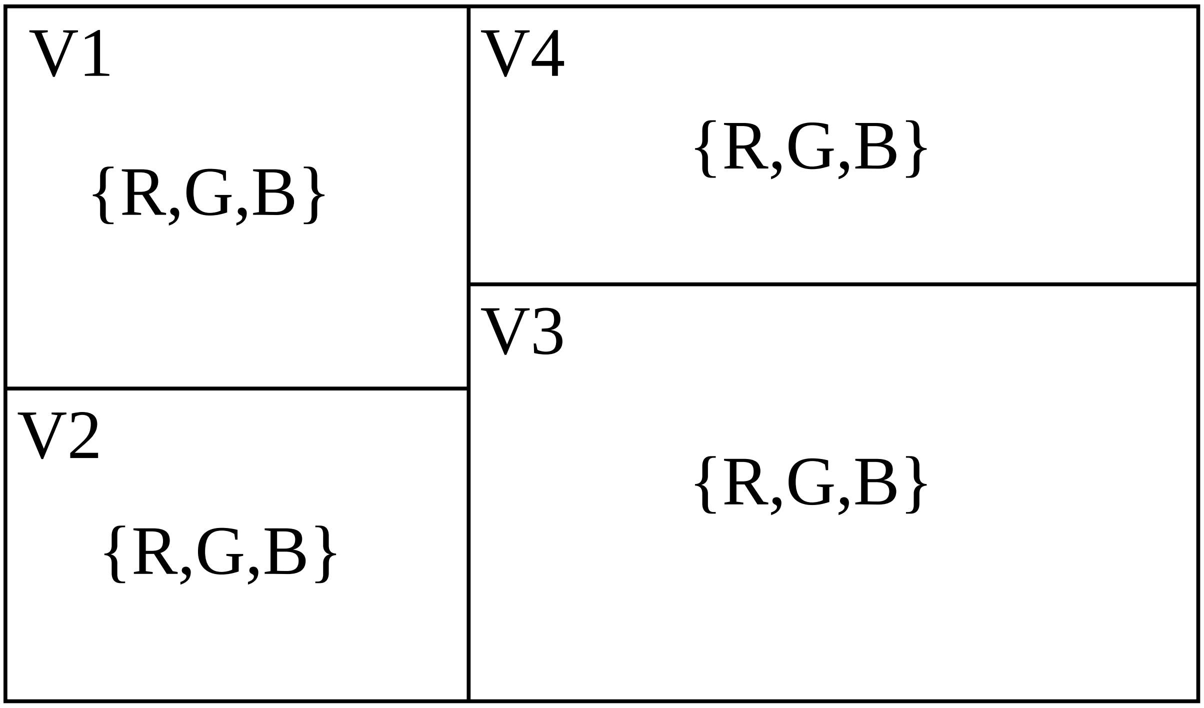
Constraint satisfaction is a computational paradigm in which knowledge is formulated as a set of *constraints* on *variables*.

The role of constraint satisfaction algorithms is to assign values to the variables in a way that is consistent with all the constraints, or to determine that no such constraint exists.

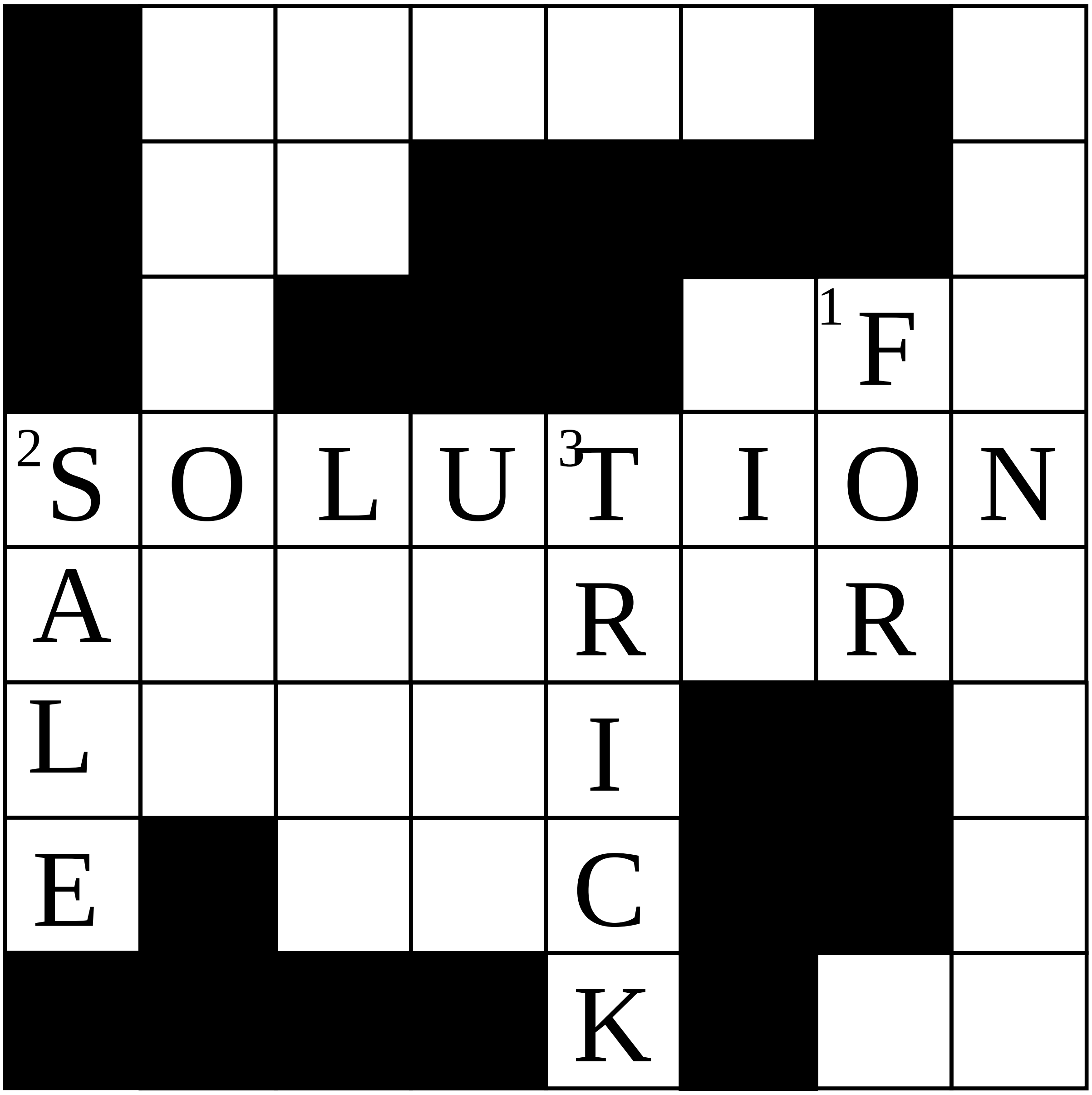
Classes of problems whose solution involves constraint satisfaction include:

- Map Colouring: variables are *regions*.
- Crossword Puzzles: variables are *clues*.
- Machine Vision: variables are *edges* in line drawings.
- Temporal Reasoning: variables are pairs of *time imtervals*.
- Belief Maintenance: variables are *beliefs*.

Map Colouring Problems



Crossword Puzzles

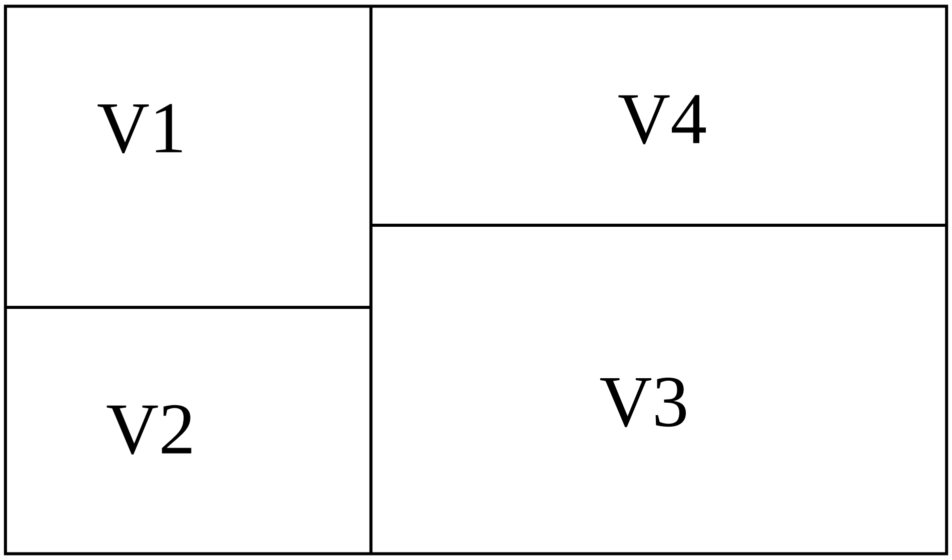


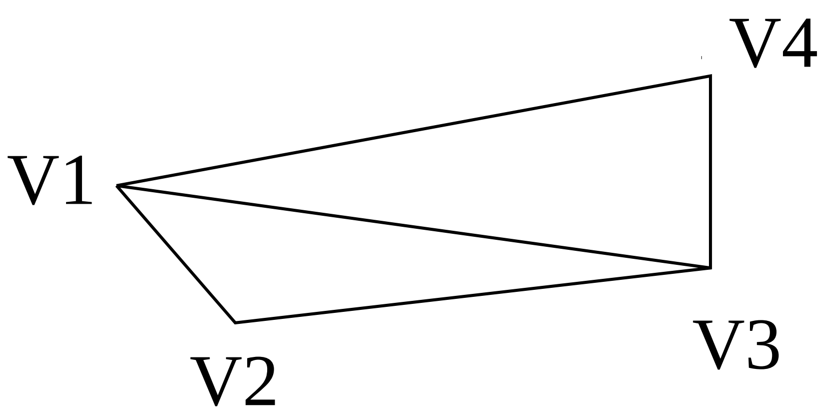
Constraint Satisfaction Problems

A **constraint network** comprises:

- A finite set of variables $V = \{V_1, \ldots, V_n\}$.
- A domain D_i for each variable V_i.
- A set of constraints $\{C_1, \ldots, C_m\}$.
- Each constraint is defined over some subset of the orginal set of variables and limits the combinations of values that variables in the subset can take.
- The goal is to find assignments to the variables that satisfy all the constraints.

Map Colouring: Example of A CSP





Map Colouring Problem

- Variables $\{V_1, V_2, V_3, V_4\}$.
- Domain $= \{R, G, B\}$ for each V_i.
- Constraints:

  $\{V_1 \neq V_2, V_1 \neq V_3, V_1 \neq V_4, V_2 \neq V_3, V_3 \neq V_4\}$
- All constraints in the list are binary (in this problem)
- In the general case n-ary contraints are permitted.
- Of particular interest also are unary constraints that limit the possible values of individual variables.

Crossword Problem

- Variables $\{D_1, A_2, D_2, D_3\}$.
- Domain = {W | W is an English word}
- Constraints:
  - Unary Constraints: value must match length of clue.

    length(D_1)=3
    length(A_2)=8
    length(D_2)=4
    length(D_3)=5
  - Binary Constraints: intersecting clues must share a character.

    $D_1[1] = A_2[7]$
    $A_2[1] = D_2[1]$
    $A_2[5] = D_3[1]$
    $D_1[2] = A_2[7]$

Constraint Satisfaction Techniques

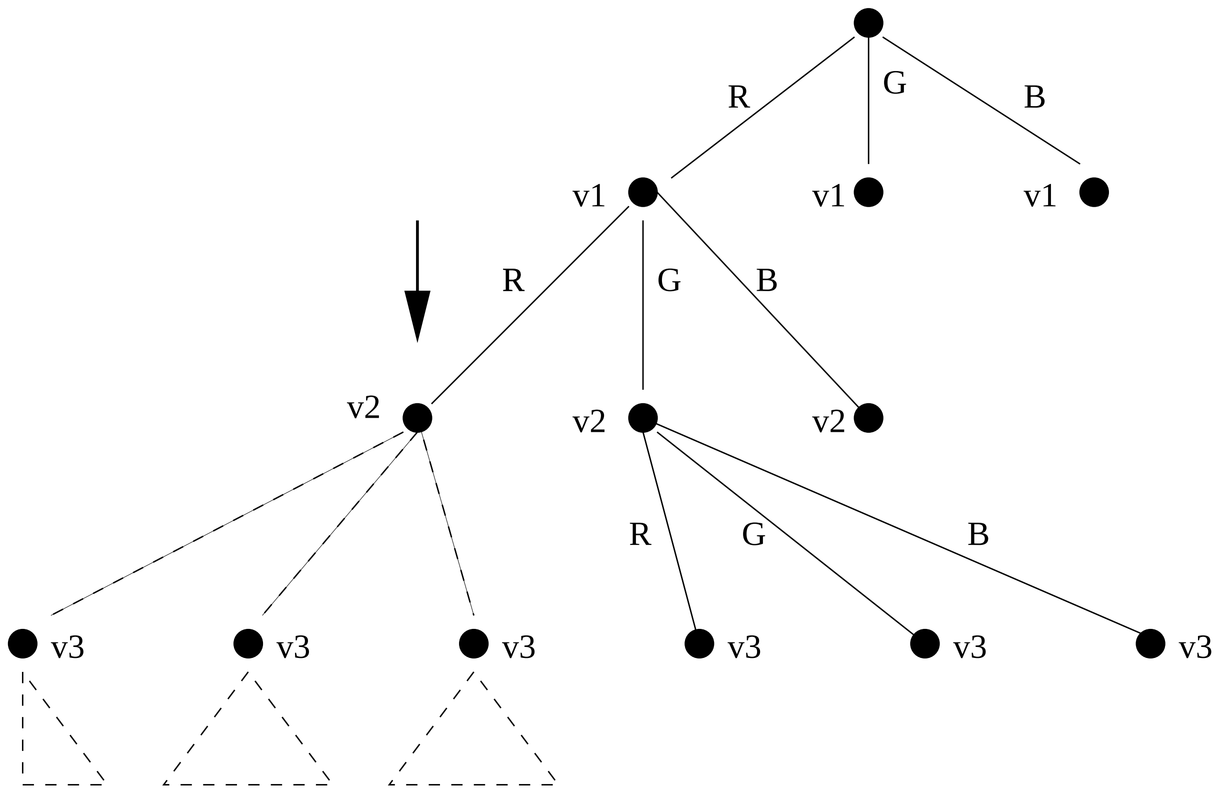
- **Generate and Test:** Each possible combination of the variables is systematically generated and tested to see if it satisfies the constraints. Problem: the number of possible combinations is equal to the Cartesian product of all the variable domains.

Variable	**Domain**
V1	{R,G,B}
V2	{R,G,B}
V3	{R,G,B}
V4	{R,G,B}

  For a problem in N variables with average domain size S, the size of the search space is clearly S^N.

- **Backtracking:** Variables are instantiated sequentially. If a constraint is violated backtrack to the most recently instantiated variable that still has alternatives.

Backtracking Offers a Slight Improvement



Backtracking is a slight improvement over generate and test in the sense that certain parts of the search space are never. Above failure occurs where the arrow is pointing, just after v2 has been assigned. Clearly, the entire search tree below that point (dotted lines) is never examined..

Improving the Efficiency of Backtracking

Despite the limited improvement offered by backtracking, it is inherently a last resort mechanism.

- **Variable Ordering:** The amount of backtracking may be greatly affected by the order in which variables are instantiated. Typical heuristics are based on the intuition that highly constrained variables should be instantiated first.
- **Value Ordering:** The order in which values for a particular variable are examined can also affect the amount of backtracking that ensues.
- **Backjumping:** When failure occurs, it is sometimes possible to identify a set of previous choices that led to it and backtrack directly to one of those rather than to the most recent previous choice.

Node Inconsistency

The main problem with backtracking is **thrashing**, where search in different parts of the space keeps failing for the same reason.

- The simplest cause of thrashing is *node inconsistency*, i.e. when the domain of a variable V contains a value that violates some unary constraint on V. For example, in the crossword puzzle domain, each variable (i.e. clue) is associated with a length L and each permissible value is constrained to be of length L.

- The obvious way to deal with node inconsistency is to **remove inconsistent values from the domains of individual variables**. Consequently, fewer bad choices of variable values are made and less backtracking occurs.

Arc Inconsistency

- A second cause of thrashing is *arc inconsistency*, when the domain of a variable V_i contains a value which is inconsistent with at least one of the values in the domain of another variable V_j. Suppose the variables were instantiated in the order

$$V_1, \ldots, V_i, \ldots, V_k, \ldots, V_j, \ldots, V_n$$

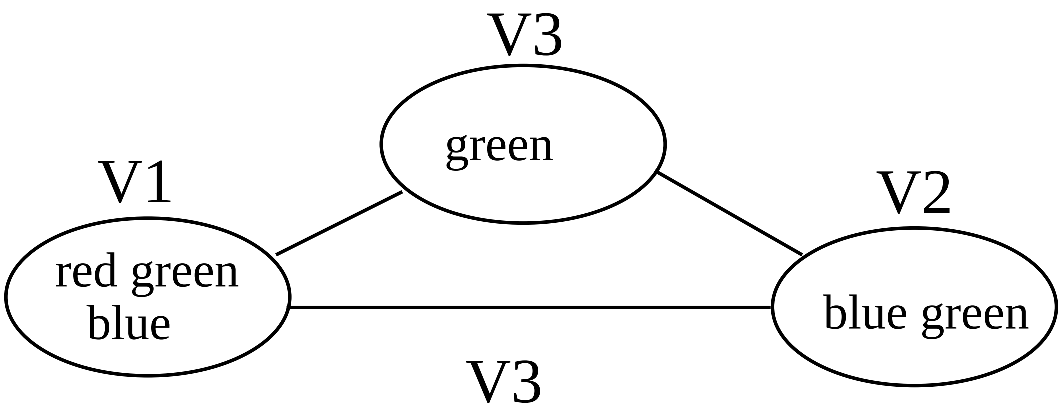
  and suppose that whenever V_i takes on the value v, it is inconsistent with **all** values in the domain of V_j. Clearly the search will fail for every choice amongst the values of variables V_k $i < k < j$, all of which have to be explored through backtracking.

- Thrashing because of arc inconsistency can be avoided if each arc (V_i, V_j) of the constraint graph is made consistent before any search takes place.

Arc Consistency

Arc (V_i, V_j) is *arc consistent* if for every value x in the current domain of V_i there is some value y in the domain of V_j such that $V_i = x$ and $V_j = y$ is permitted by the binary constraint between V_i and V_j.

Note that the concept of arc consistency is *directional*: if (V_i, V_j) is consistent, this does not automatically mean that (V_j, V_i) is consistent.



In this graph, (V_3, V_2) is consistent because because green is the only value of the domain of V_3, and for that value, there is an assignment (V_2=blue) that satisfies the constraint.

However, arc (V_2, V_3) is inconsistent, since for V_2=green, there is no assignment in D_3 which satisfies the inequality constraint.

Enforcing Arc Consistency

Clearly an arc (V_i, V_j) can be made consistent by simply deleting those values from the domain of V_i for which the previous condition is false. Let's call this procedure REVISE.

However, to make every arc of the constraint graph consistent, it is **not** sufficient to execute REVISE for each arc just once.

If REVISE reduces the domain of some variable V_i then each previously revised arc (V_j, V_i) **must be revised again** because some members of D_j might not be compatible with the now reduced D_i.

For instance (V_1, V_2) is initially consistent, but after (V_2, V_3) is made consistent by deleting green from D_2, arc (V_1, V_2) becomes inconsistent.