Constraint Satisfaction Problems

Frank C. Langbein
⟨F.C.Langbein@cs.cf.ac.uk⟩

Department of Computer Science
Cardiff University

13th February 2001

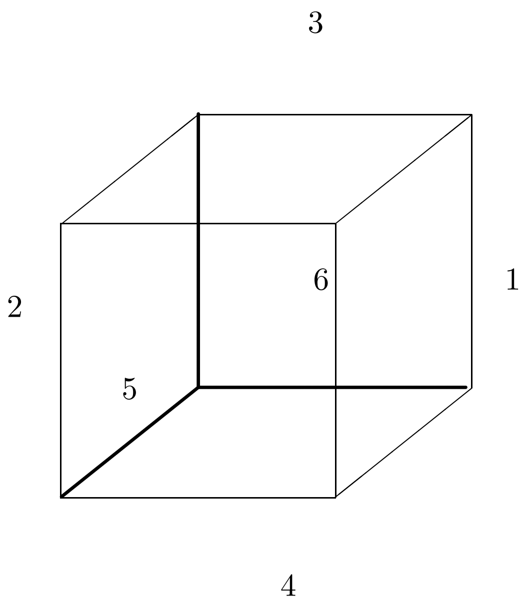
Constraint Satisfaction Problems (CSPs)

- A CSP is a high level description of a problem.
- The model for the problem is represented by a set of variables and their domain.
- The problem is stated as constraints specifying the relations between the variables.
- The constraints only specify the relationships *without* specifying a computational procedure to enforce that relationship.
- The computer has to find a solution to the specified problem.

Applications

- Interpreting objects in 3D scenes.
  Scene labelling.
- Solid Modelling.
  Constrained–based design, Beautification.
- Advanced Planning and Scheduling.
  Well–activity scheduling, production scheduling.
- Assignment problems.
  Stand allocation for airports, balancing work among different persons.
- Network management and Configuration.
  Planning of cabling of telecommunication networks, network reconfiguration without service interruptions.
- Database systems.
  Ensure and/or restore data consistency.
- Molecular biology.
  DNA sequencing, chemical hypothesis reasoning, protein docking.
- Electrical engineering.
  Fault location, circuit layout computation.

A Geometric CSP



- Planes: ${n_l}^t x = p_l$, $l = 1, \ldots, 6$
- Orthogonal planes: $n_1^t n_3 = 0$, $n_1^t n_4 = 0$, $\ldots$
- Parallel planes: $n_1^t n_2 = 1$, $\ldots$

Elements of the CSP:

- Variables: $n_l \in S^2$ (unit sphere), $p_l \in \mathbb{R}$
- Domains: S^2, $\mathbb{R}$
- Constraints:
  - Plane l and k orthogonal:
    Variables: (n_l, n_k)
    Valid set: $S^2 \times S^2 \setminus \{(n_l, n_k) \in S^2 \times S^2 : n_l^t n_k \neq 0\}$
  - Plane l and k parallel:
    Variables: (n_l, n_k)
    Valid set: $S^2 \times S^2 \setminus \{(n_l, n_k) \in S^2 \times S^2 : n_l^t n_k \neq 1\}$

Constraint Satisfaction Problem

Definition:
An **instance of a CSP** is a triple (X, D, C), where

- X is a (finite) set of variables,
- D is the domain for the variables,
- C is a set of constraints $\{C_1, C_2, \ldots, C_n\}$.
  Each constraint C_l is a pair (s_l, R_l), where
  - $s_l = (x_{l_1}, \ldots, x_{l_m})$ is an m–tuple of variables (scope),
  - R_l is an m–ary relation over D, i.e.
    R_l is a subset of all possible variable values representing the allowed combinations of simultaneous values for the variables in s_l.

A **solution of an instance of a CSP** is a function $f : X \to D$, such that

$$\forall (s_l, R_l) \text{ with } s_l = (x_{l_1}, \ldots, x_{l_m}) \quad (f(x_{l_1}), \ldots, f(x_{l_m})) \in R_l.$$

Sometimes $f(X)$ is called the solution.

Note: In general each variable can have its own domain.

Generalised Constraint Satisfaction Problem

Relational Structure: $\Sigma = \langle X, E_1, \ldots, E_q \rangle$
– X is a non–empty set (universe)
– E_l is a relation over X

Example: A graph is a relational structure where the universe is the vertex set and a single relation specifying which vertices are adjacent.

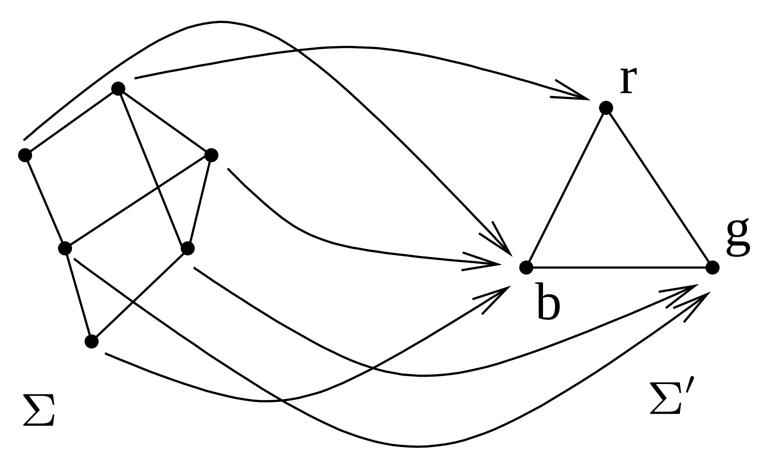
Homomorphism: $h \in \mathsf{Hom}(\Sigma, \Sigma') :\Leftrightarrow h : X \to X'$
such that for all $l = 1, \ldots, q$
$(x_1, \ldots, x_m) \in E_l \Rightarrow (h(x_1), \ldots, h(x_m)) \in E'_l$

The solutions of a CSP (X, D, C) with $C = \{(s_1, R_1), \ldots, (s_n, R_n)\}$ are equal to the homomorphisms between

$$\Sigma = \langle X, \{s_1\}, \ldots, \{s_n\} \rangle, \quad \Sigma' = \langle D, R_1, \ldots, R_n \rangle.$$

Generalized CSP: Find homomorphisms between two relational structures $\langle X, E_1, \ldots, E_n \rangle$, $\langle D, R_1, \ldots, R_n \rangle$, where E_l and R_l have the same *arity* (for all l).

Example: Graph Colourability



$h \in \mathsf{Hom}(\Sigma, \Sigma')$

Properties of a CSP

- Solving CSPs is in general NP–complete.
  Identification of restrictions that make the problem tractable is very important.
- Each CSP can be converted into a binary CSP.
- Overconstrained CSP.
  The CSP contains more constraints than required which may be inconsistent and/or redundant.
- Underconstrained CSP.
  The CSP cannot be solved uniquely.
- Expressive power of a set of constraint types.

Solving CSPs

Solving a CSP could mean to find

- one solution, without preference as to which one,
- all solutions,
- an optimal, or at least a good solution.

General methods for solving a CSP:

- Combinatorial methods for finite D.
  Solutions can be found by systematic search in D:
  - Traverse the space of partial solutions.
  - Explore the space of complete value assignments.
- Analytical methods for infinite D.
  Solutions can be found by analysing the constraints as some (generalised) equation system:
  - Solve the constraints simultaneously.
  - Consider the constraints sequentially.

Systematic Search

Generate and Test: Assign a value to each variable.
If it is a solution, stop.
Otherwise modify the assignment.

Searches all of D.

Improvements:

- Use an informed/smart generator such that the conflicts found by the tester are minimised.
  $\longrightarrow$ stochastic algorithms

- Merge the generator with the tester.
  $\longrightarrow$ backtracking

Backtracking

Backtracking: Instantiate the variables sequentially.
Test the validity of a constraint as soon as its respective variables are instantiated.
If a constraint is violated, backtrack to the most recently instantiated variable for which there are still values left.

Whenever a constraint is violated a complete subspace of D is eliminated.

Problems:

- Trashing: repeated failure due to the same reason generated by older variable assignments.
- Redundancy: rediscovering of the same inconsistencies.

$\longrightarrow$ Heuristics for variable ordering:
– Assign the variable with the fewest possible remaining alternatives first.
– Instantiate the variables first that participate in the highest number of constraints.

$\longrightarrow$ Dependency directed backtracking:
– Inconsistencies are noted whenever they are detected.
– Avoids trashing and redundancy.
– Even if the search space is minimal, detecting inconsistencies and choosing new values is quite complex.

Consistency Techniques

Principle: Remove inconsistent values from the variable domains until a solution is found.

The CSP is represented as a graph using binary constraints only.

K–consistency: For every system of values for $K-1$ variables satisfying all the constraints between them, there exists a value for an arbitrary K–th variable such that all constraints between the K variables are satisfied.

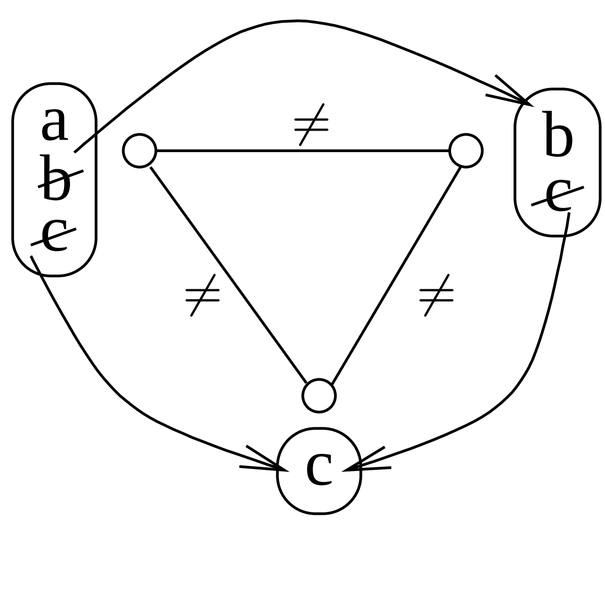
A constraint graph is *strongly K–consistent* if it is J–consistent for all $J \leq K$.

$\longrightarrow$ For a strongly N–consistent graph with N nodes a solution can be found without searching.

$\longrightarrow$ Obtaining N–consistency in a graph with N nodes is exponential.

$\longrightarrow$ A strongly K–consistent graph with $N > K$ nodes still requires searching (backtracking).

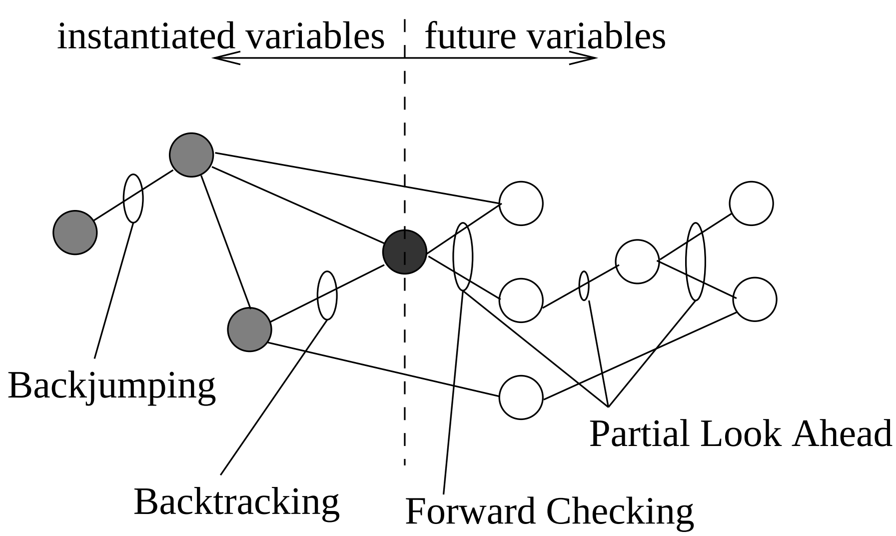
Example: strong 2–consistency (arc–consistency)



Constraint Propagation

Principle: Combine backtracking with consistency techniques.

- **Look Back**:
  Consistency checks among already instantiated variables.
  - Analyse the situation in order to find the source of the inconsistency and backtrack to the most recent conflicting variable (backjumping).
  - Remember incompatible assignments of variables (backchecking/backmarking).
- **Look Ahead**:
  Prevent future conflicts by limiting the domains of uninstantiated variables.
  - When a value is assigned (temporarily) remove any value of a *future* variable which conflicts with this assignment (forward checking).
  - In addition remove values of variables indirectly depending on the instantiated variable (partial look ahead).
  - After each assignment a full consistency check on the graph is performed (full look ahead).



Stochastic and Heuristic Methods

- Instantiate all variables randomly.
- Use a *repair* or *hill–climbing* metaphor to move towards more and more complete solutions.
- Stop if a complete solution is found.

Update strategies:

Hill–climbing:	–	Modify the value of one variable such that more constraints are satisfied.
	–	Restart with a random assignment if no more constraints can be satisfied (local minimum).
Min–conflicts:	–	Randomly choose a variable that is involved with an unsatisfied constraint.
	–	Pick a value that reduces the number of unsatisfied constraints.
	–	If no such value exists pick a random value which does not increase the number of unsatisfied constraints.
Random walk:	–	Select a variable with a probability p and apply min–conflicts or hill–climbing with probability $1 - p$.
Tabu search:	–	Keep a list of recent configurations which are (temporarily) tabu.
	–	Tabu restrictions may be overridden under certain conditions (aspiration criteria).

CSPs with infinite Domains as Equation Systems

For $D = \mathbb{R}$, the CSP can be represented as an equation system:

$$f_1(x) = 0$$
$$\vdots$$
$$f_n(x) = 0$$

with $f_l : \mathbb{R}^n \longrightarrow \mathbb{R}$.

Symbolic Solver

- Reliable methods available (Gröbner basis techniques).
- Identifies inconsistencies.
- Expensive, only suitable for small CSPs.

Numerical solver

- Equation system solver (Homotopy methods, Newton–Raphson).
- Optimization methods.
  - Minimize an error function, like least squares error.
  - Naturally handles inconsistencies, but generates *average solution*.
  - Suitable for larger CSPs.
  - Problems caused by *bad* objective functions: slow/no convergence, local minima.
  - Methods:
    * Quasi–Newton methods (BFGS).
    * Gauss–Newton / Trust region methods (Levenberg–Marquardt).
    * Hybrids between Quasi–Newton and trust region.
    * Evolutionary methods.

Other methods for Infinite Domains

Local Propagation

- Repeatedly select uniquely satisfiable constraints.
- A single constraint determines the value for a variable.
- Once a variable value is known, another constraint might be solvable.
- An initial planning phase to choose the order of the constraints is required.
- Restrictions:
  - Most algorithms solve equality constraints only.
  - Cyclic constraints cannot be solved.

Decomposition

- Partition the constraint graph into (vertex–induced dense) subgraphs.
- A subgraph corresponds to a subproblem which can be solved separately.
- General strategy:
  - Find a suitable subgraph.
  - Solve the subgraph problem.
  - Reduce the graph by replacing the subgraph by a single node.
  - Find the next subgraph.

Constraint Optimization

Find a solution to a CSP which also minimizes an objective function mapping every solution to a numerical value.

Branch and Bound (for finite D):

- Backtracking algorithm.
- For each partial solution the objective function is approximated (underestimated).
- If the estimation for a partial solution exceeds some bound, the complete subspace is removed.
- Initially the bound is $+\infty$ and it is set to the value of the objective function for the best solution found so far.

Constrained Numerical Optimization (for infinite D):

- Methods for linear constraints and special objective functions (Simplex method, Goldfarb Idnani method, . . .).
- Gradient projection and reduction methods.
- Penalty and multiplicator methods.
- Sequential Quadratic Approximation (SQP).
- Statistical methods.

Inconsistent Constraints

Not all constraints in the constraint set can be satisfied simultaneously.

$\longrightarrow$ A solution should satisfy a subset containing the *important* constraints.

Partial Constraint Satisfaction (finite D):

- Find values of a subset of variables that satisfy a subset of constraints.
- Some constraints are weakened to permit additional value combinations.
- The goal is to find a solution with the best value of some function evaluating the solution.

Constraint Hierarchies (infinite D):

- Constraints are weakened explicitly by specifying a strength or preference.
- Weaker constraints are not allowed to break a stronger constraint.
- *Refining methods*: Start with satisfying constraints on the strongest level and continue with weaker levels.
- *Local propagation*.

Intelligent Constraint Hierarchies:

- Identify as many inconsistent constraints as possible and start with a *good* set of consistent constraints.
- Solve the constraint system using
  - a numerical optimization method with weighted constraints,
  - a local propagation method.
- Use some reasoning to detect further inconsistencies between the constraints from the results of the solver and note those using a belief network or similar.
- Depending on the change in the belief network modify the constraint system.