An Introduction to Constraint Programming

Barbara M. Smith

University of Huddersfield

Constraint Satisfaction Problems

A constraint satisfaction problem (CSP) consists of:

- a set of *variables* $\{x_1, ..., x_n\}$;
- for each variable, a finite set D_i of possible values (its *domain*);
- a set of *constraints* restricting the values that the variables can simultaneously take

Constraint Satisfaction Problems

A constraint satisfaction problem (CSP) consists of:

- a set of *variables* $\{x_1, ..., x_n\}$;
- for each variable, a finite set D_i of possible values (its *domain*);
- a set of *constraints* restricting the values that the variables can simultaneously take

A constraint $C_{ijk...}$ on x_i, x_j, x_k,... is a subset of the possible combinations of values of $x_i, x_j, x_k, ...$ i.e. $C_{ijk...} \subseteq D_i \times D_j \times D_k \times .....$

Constraint Satisfaction Problems

A constraint satisfaction problem (CSP) consists of:

- a set of *variables* $\{x_1, ..., x_n\}$;
- for each variable, a finite set D_i of possible values (its *domain*);
- a set of *constraints* restricting the values that the variables can simultaneously take

A constraint $C_{ijk...}$ on x_i, x_j, x_k,... is a subset of the possible combinations of values of $x_i, x_j, x_k, ...$ i.e. $C_{ijk...} \subseteq D_i \times D_j \times D_k \times .....$

The domain of a variable is often a set of integers, or an enumerated set of values (e.g. $\{$red, green, blue$\}$ but need not be

- e.g. *set* variables have values which are sets

Expressing Constraints

- A constraint can be specified
  - *intensionally*, e.g. $x_1 \neq x_2$
  - or *extensionally* as a set of allowed tuples of values, e.g. $\{(1,2),(2,1)\}$

Expressing Constraints

- A constraint can be specified
  - *intensionally*, e.g. $x_1 \neq x_2$
  - or *extensionally* as a set of allowed tuples of values, e.g. $\{(1,2),(2,1)\}$
- In early CSP research, constraints were usually expressed extensionally - algorithms assumed extensional representation
- Now, users of CP tools almost always express constraints intensionally

Expressing Constraints

- A constraint can be specified
  - *intensionally*, e.g. $x_1 \neq x_2$
  - or *extensionally* as a set of allowed tuples of values, e.g. $\{(1,2),(2,1)\}$
- In early CSP research, constraints were usually expressed extensionally - algorithms assumed extensional representation
- Now, users of CP tools almost always express constraints intensionally
- In theory, the intensional and extensional representations of a constraint are equivalent: in practice they are not
  - e.g. We can form a single constraint as the conjunction of all the constraints of a CSP: we can easily write this intensionally, but finding the satisfying tuples is equivalent to finding all the solutions of the CSP (so NP-complete)

Examples of Constraints on Integer Variables

- $\sum_i a_{ij} x_i \leq b_j$, ($a_{ij}, b_j$ constants, x_i variable)
  - linear inequalities
- $xy = z$ where x, y, z are variables
  - arithmetic expressions involving variables and constants
- if $x_i = 1$ then $x_{i+1} = 1$
  - logical constraints can express the logic of the problem directly
- $t = \sum_i a_i x_i$ (t, a_i, x_i constants or variables)
  - constraints on arrays of variables
- allDifferent($x_1, x_2, \ldots, x_n$)
  - global constraints, affecting any number of variables

Aims in Solving CSPs

A solution to a CSP is an assignment to each variable of a value from its domain, such that all the constraints are satisfied. We might want to find:

- whether a CSP has a solution or not;
- any solution (and we don't mind which)
- all solutions
- an optimal solution, given some objective

Aims in Solving CSPs

A solution to a CSP is an assignment to each variable of a value from its domain, such that all the constraints are satisfied. We might want to find:

- whether a CSP has a solution or not;
  - we can exclude any solution, if we don't exclude them all
- any solution (and we don't mind which)
- all solutions
- an optimal solution, given some objective

Aims in Solving CSPs

A solution to a CSP is an assignment to each variable of a value from its domain, such that all the constraints are satisfied. We might want to find:

- whether a CSP has a solution or not;
  - we can exclude any solution, if we don't exclude them all
- any solution (and we don't mind which)
  - symmetry might not have much effect on how easy it is to find one
- all solutions

- an optimal solution, given some objective

Aims in Solving CSPs

A solution to a CSP is an assignment to each variable of a value from its domain, such that all the constraints are satisfied. We might want to find:

- whether a CSP has a solution or not;
  - we can exclude any solution, if we don't exclude them all
- any solution (and we don't mind which)
  - symmetry might not have much affect on how easy it is to find one
- all solutions
  - we probably want to exclude symmetrically equivalent solutions
- an optimal solution, given some objective

Solving CSPs

Constraint programming tools (e.g. ILOG Solver, Eclipse, Sicstus Prolog, ...) solve CSPs by a combination of:

- systematic search through the space of possible assignments of values to variables
  - choose a variable, *var*
  - assign a value to it, *val*
  - backtrack to try another choice if a failure occurs
  - repeat if not
- constraint propagation: use the constraints to derive new information about the problem, both initially and from the new constraint *var* = *val*
  - a failure occurs when some future variable has no remaining values, following constraint propagation

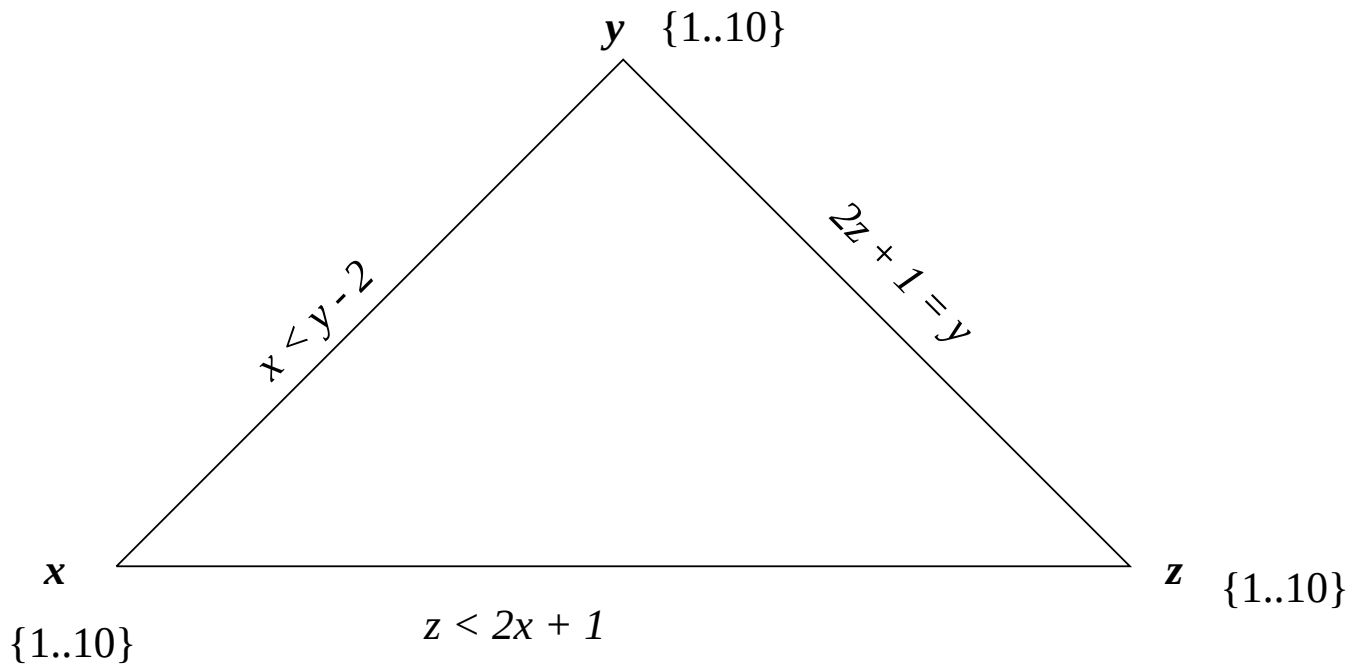
Other Solution Methods

- CSPs can also be solved by incomplete methods e.g. stochastic local search
- It is not clear that symmetry makes solving CSPs harder for these methods
- e.g. Steve Prestwich has shown that removing symmetry can be harmful
  - he has suggested deliberately *increasing* the symmetry in a problem to help stochastic local search find a solution

Constraint Propagation: Binary CSPs

If all the constraints of a CSP are binary, the variables and constraints can be represented in a *constraint graph*:

- nodes represent variables
- edges represent constraints

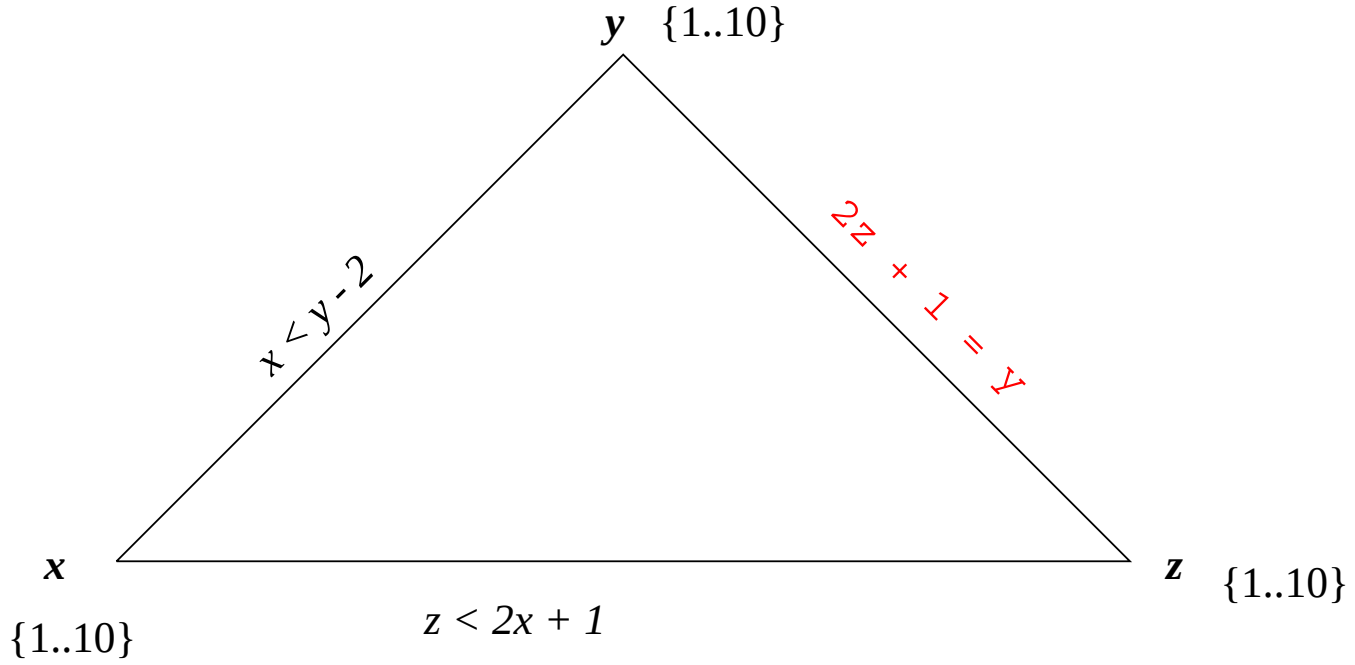


Constraint Propagation: Binary CSPs

If all the constraints of a CSP are binary, the variables and constraints can be represented in a *constraint graph*:

- nodes represent variables
- edges represent constraints

By considering each arc of the graph in turn, we may be able to remove values from the domains of the variables:

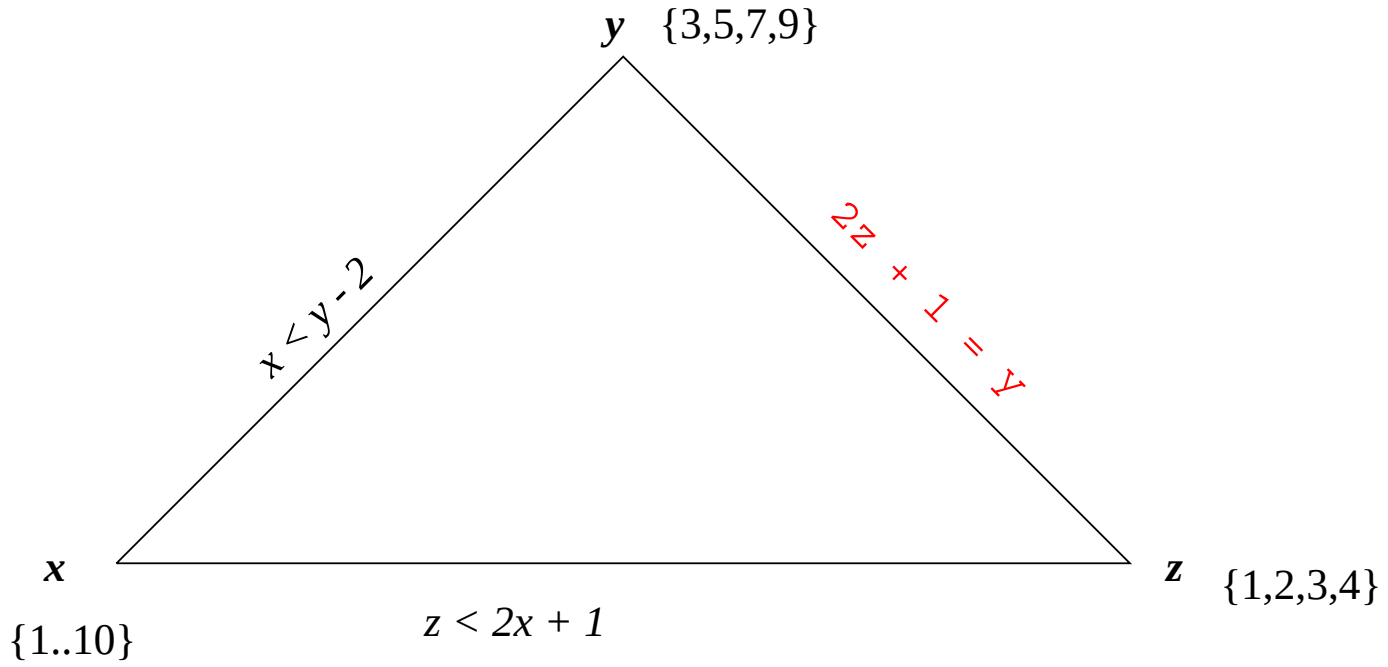


Constraint Propagation: Binary CSPs

If all the constraints of a CSP are binary, the variables and constraints can be represented in a *constraint graph*:

- nodes represent variables
- edges represent constraints

By considering each arc of the graph in turn, we may be able to remove values from the domains of the variables:

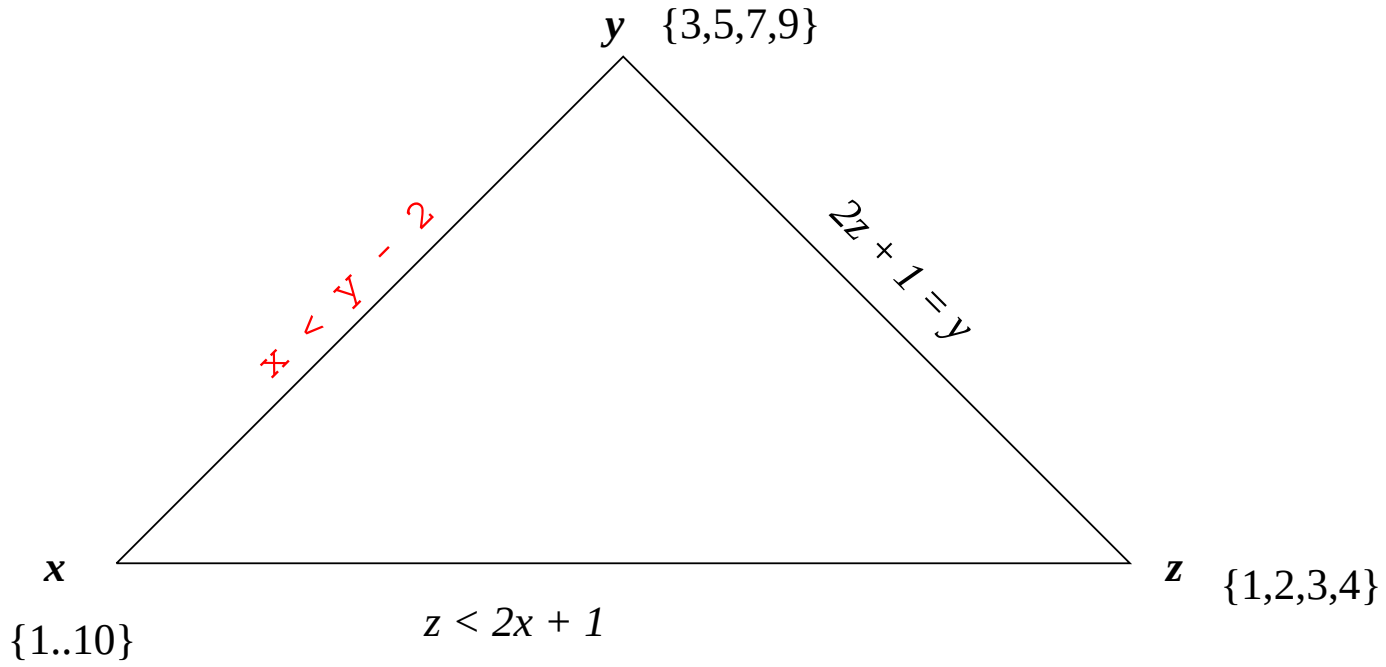


Constraint Propagation: Binary CSPs

If all the constraints of a CSP are binary, the variables and constraints can be represented in a *constraint graph*:

- nodes represent variables
- edges represent constraints

By considering each arc of the graph in turn, we may be able to remove values from the domains of the variables:

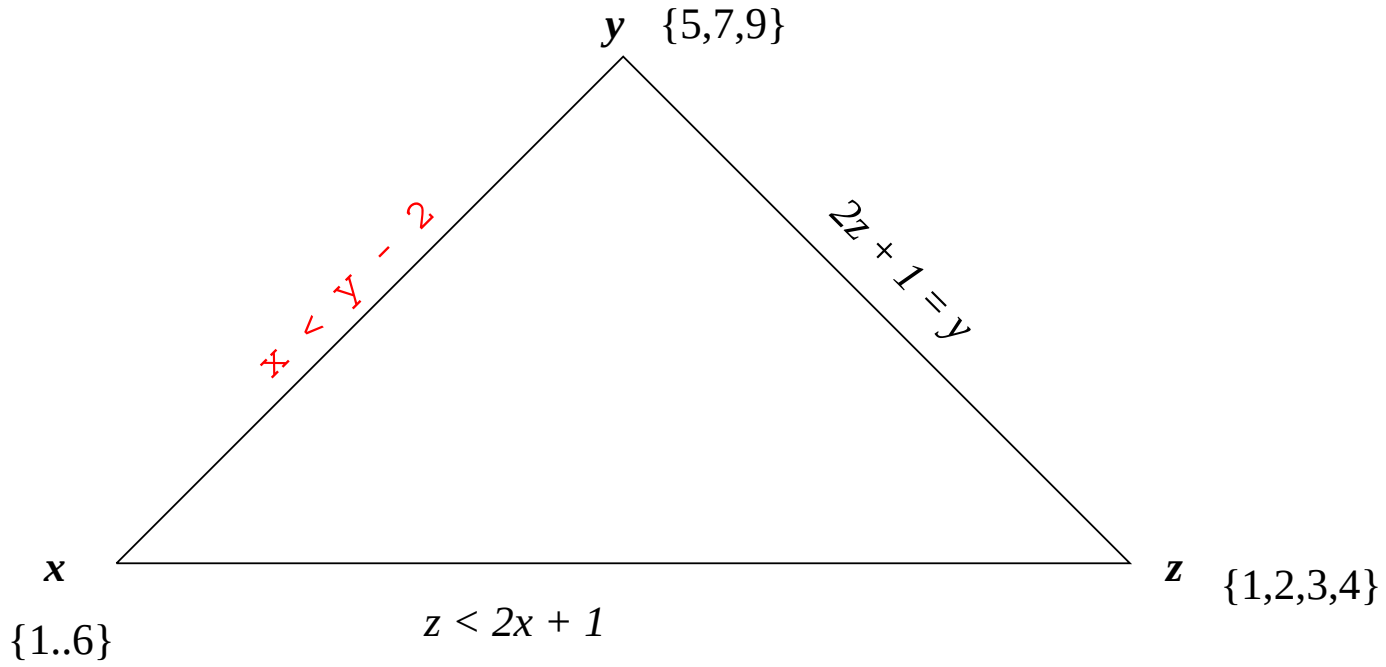


Constraint Propagation: Binary CSPs

If all the constraints of a CSP are binary, the variables and constraints can be represented in a *constraint graph*:

- nodes represent variables
- edges represent constraints

By considering each arc of the graph in turn, we may be able to remove values from the domains of the variables:

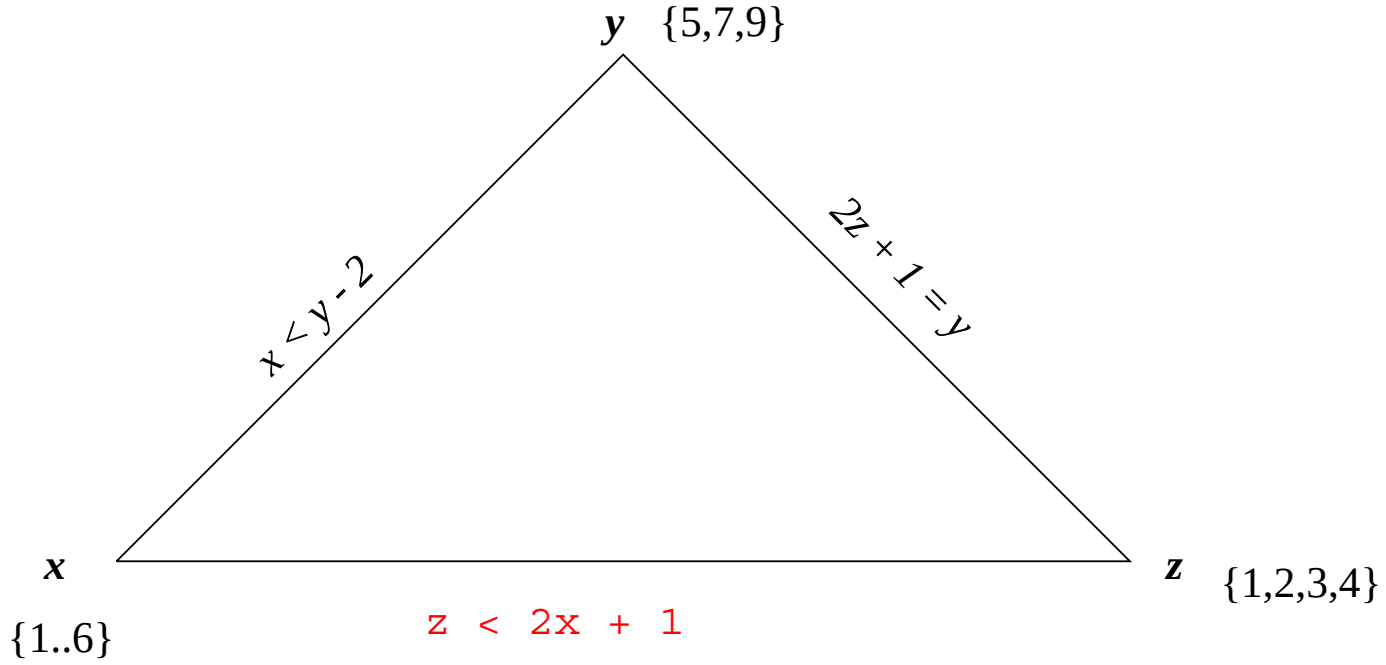


Constraint Propagation: Binary CSPs

If all the constraints of a CSP are binary, the variables and constraints can be represented in a *constraint graph*:

- nodes represent variables
- edges represent constraints

By considering each arc of the graph in turn, we may be able to remove values from the domains of the variables:

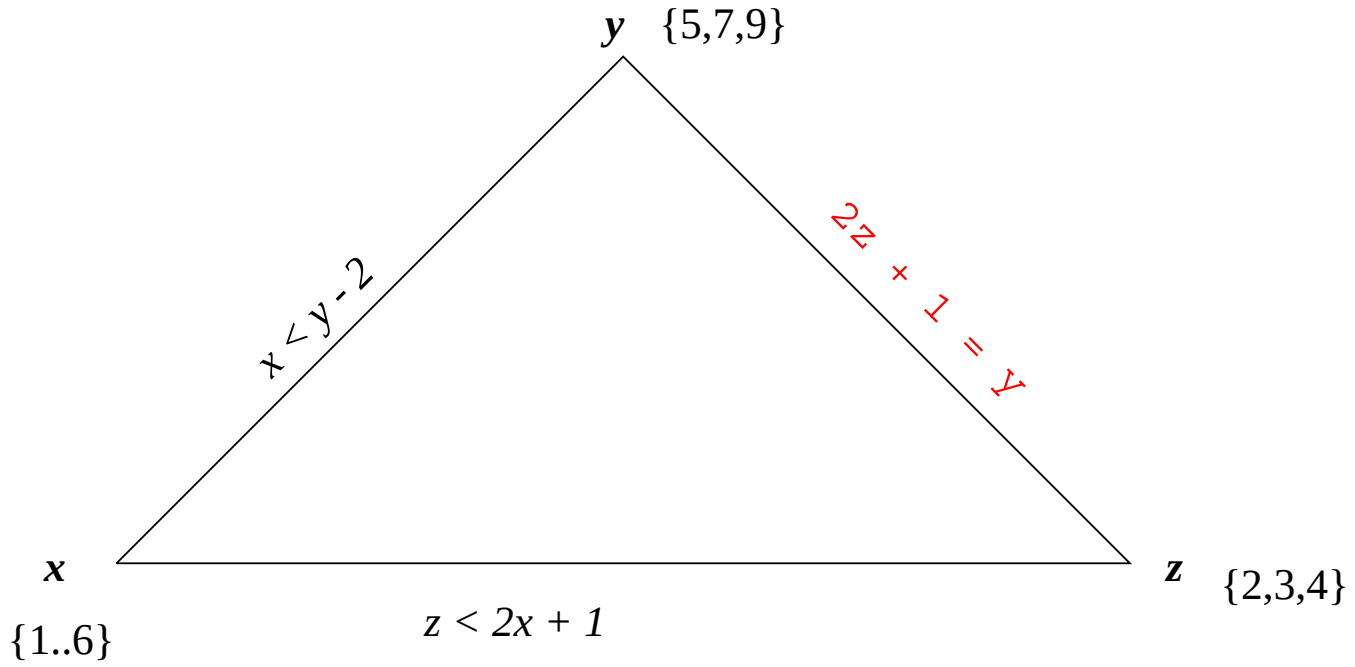


Constraint Propagation: Binary CSPs

If all the constraints of a CSP are binary, the variables and constraints can be represented in a *constraint graph*:

- nodes represent variables
- edges represent constraints

By considering each arc of the graph in turn, we may be able to remove values from the domains of the variables:



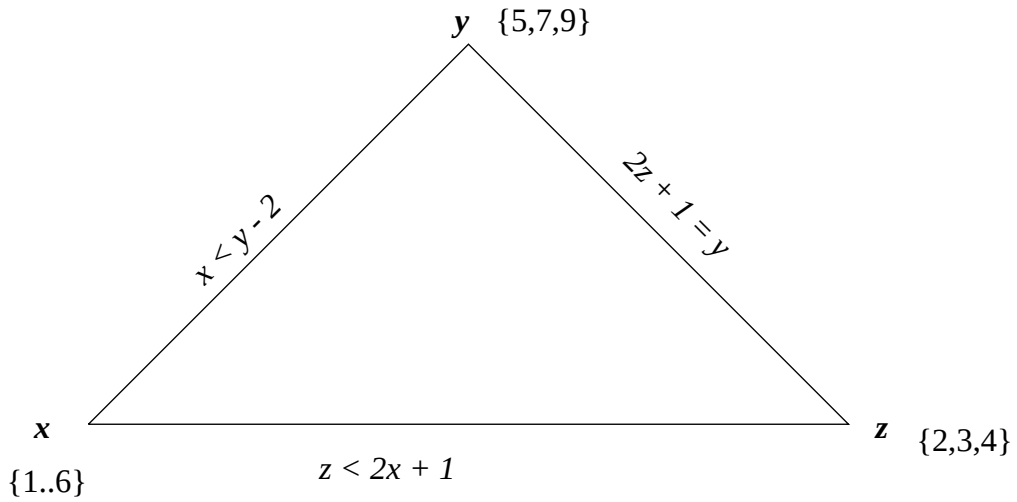
Arc Consistency

- The arc (x_i, x_j) is arc consistent if for every value $a \in D_i$, there is a value $b \in D_j$ such that the assignments $x_i = a$ and $x_j = b$ satisfy the constraint C_{ij}
- The value $b \in D_j$ supports a
- We can delete any value in D_i unless it has at least one supporting value in the domain of every variable which constrains x_i
- If every arc in the constraint network is arc consistent, the problem is arc consistent
- Enforcing AC on binary constraints is fast, especially with algorithms tailored for specific types of constraint

Generalised Arc Consistency

- AC considers only binary constraints
- We can generalise it to non-binary constraints
- We consider a single variable in relation to a single constraint and remove values from the domain of the variable if they are not supported by the other variables involved in the constraint
  - Suppose there is a constraint $C_{x_1 x_2 x_3 \ldots x_k}$
  - For any value $a_1 \in D_1$ there must be a set of values $a_2, a_3, \ldots, a_k$ for the variables $x_2, x_3, \ldots, x_k$ such that $(a_1, a_2, a_3 \ldots, a_k)$ satisfies $C_{x_1 x_2 x_3 \ldots x_k}$
  - If no set of values can be found, a_1 can be removed from the domain of x_1
- General algorithms for achieving generalised arc consistency are too expensive to apply, but there are faster algorithms for specific types of constraint, e.g. the allDifferent constraint

Bounds Consistency



- For some constraints, only the bounds of the variable's domains are reduced:
  - for some constraints, e.g. $x < y - 2$, this does achieve arc consistency
  - otherwise, enforcing BC is faster than enforcing AC
- Because constraint propagation is interleaved with search, it has to be fast
- There has to be a trade-off between:
  - Finding all possible domain reductions i.e. keeping every constraint GAC, and thereby reducing search as much as possible
  - Doing less domain reduction, so saving time to do more search

Enforcing More Consistency

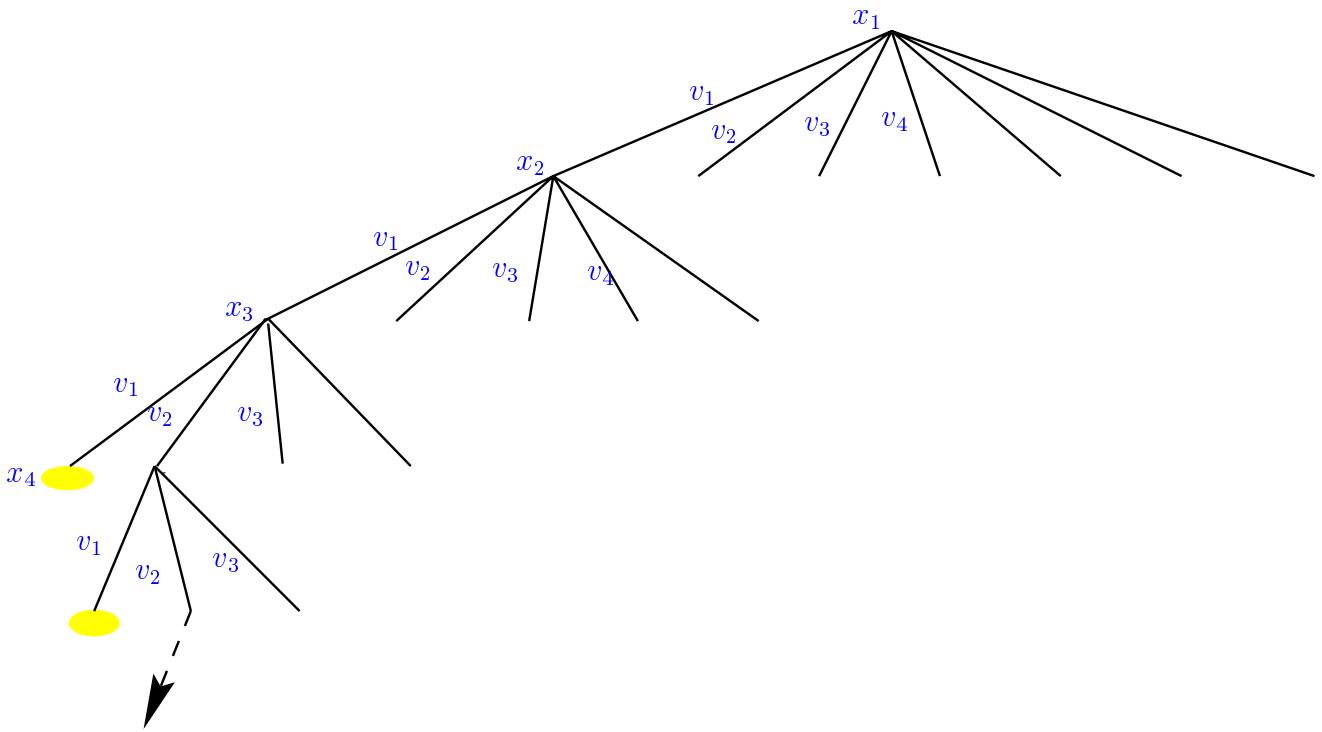
- There are consistency algorithms that consider more than one constraint together, e.g. path consistency
- Suppose
  - there are three variables x_i, x_j, x_k with binary constraints between x_j and each of x_i, x_k
  - the values $v_i \in D_i$ and $v_k \in D_k$ are allowed by the constraint C_{ik}
- If no value can be found for x_j which is simultaneously consistent with $x_i = v_i$ and $x_k = v_k$, then we cannot allow v_i and v_k to be simultaneously assigned to x_i and x_k respectively
  - But PC algorithms are slow, and assume extensional representation of constraints (we remove the pair (v_i, v_k) from the extensional representation of the constraint C_{ik})

Solving CSPs (Again)

Constraint programming tools solve CSPs by a combination of:

- systematic search through the space of possible assignments of values to variables
  - choose a variable, *var*
  - assign a value to it, *val*
  - backtrack to try another choice if a failure occurs
  - repeat if not
- constraint propagation: use the constraints to derive new information about the problem, both initially and from the new constraint *var* = *val*
  - a failure occurs when some future variable has no remaining values, following constraint propagation

Depth-First Search



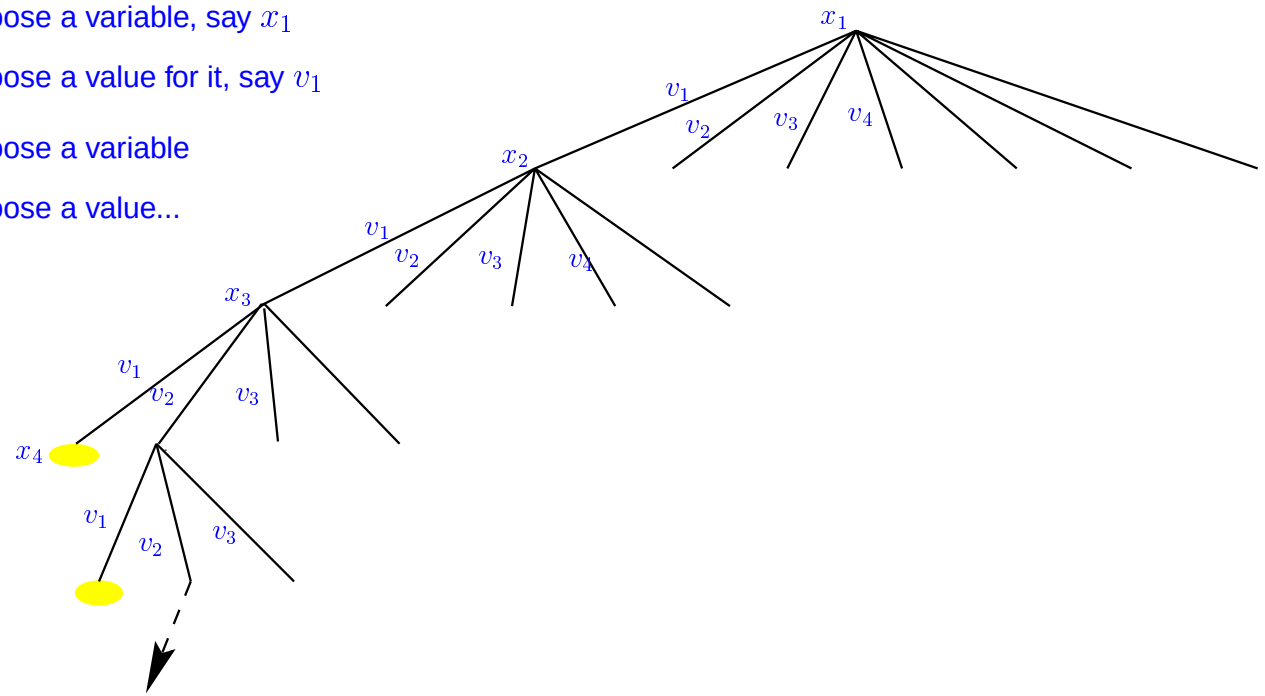
Depth-First Search

Choose a variable, say x_1

Choose a value for it, say v_1

Choose a variable

Choose a value...



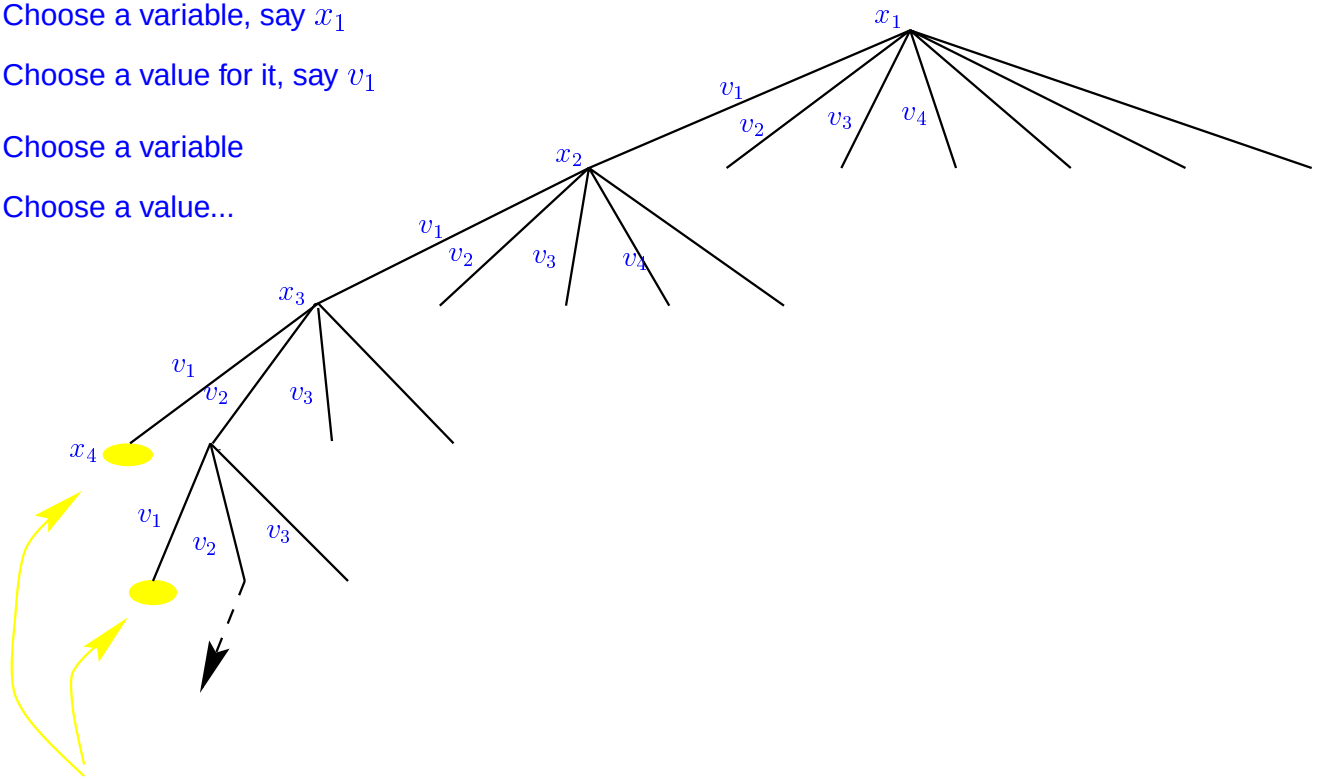
Depth-First Search

Choose a variable, say x_1

Choose a value for it, say v_1

Choose a variable

Choose a value...



If a failure is detected, backtrack

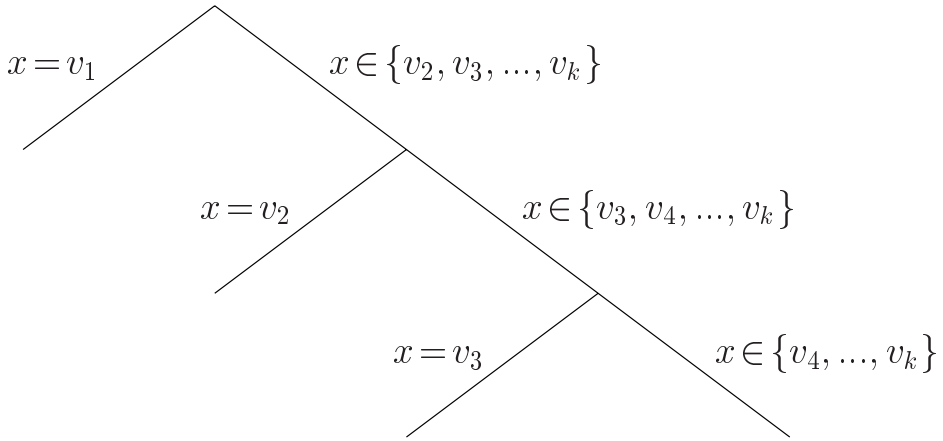
Binary Branching

- But constraint programming tools do

not build search trees like this

Binary Branching

- But constraint programming tools do not build search trees like this
- By default, they construct a sequence of binary choices:
- On the left branch, the next value is assigned
- On the right branch, this value is removed from the domain
  - This constraint is propagated before a new choice point is created



Constraint Propagation in Binary Branching

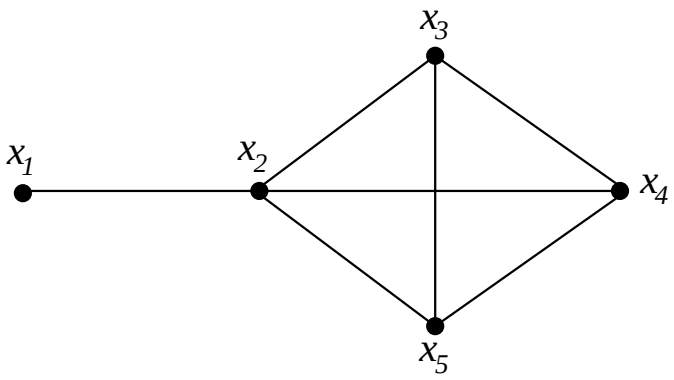
- On the left branch, the constraint $var = val$ is added
  - typically leads to a lot of constraint propagation
- On the right branch, the constraint $var \neq val$ is added
  - if var has only one remaining value, this gets assigned
  - otherwise, the value val may be the only supporting value for some value in another variable's domain
  - can trigger a sequence of domain reductions
  - occasionally, all the remaining values of var are removed, and the right branch terminates

Other Forms of Binary Branching

- Even if var has two or more values on the right branch (after constraint propagation), the next choice point need not involve var
  - i.e. each choice point can choose *any* variable from those still unassigned
- Other types of binary choice can be made
  - e.g. domain splitting: $var < m$ on the left branch, $var \geq m$ on the right branch, where m is a value in the middle of the domain

Variable Ordering

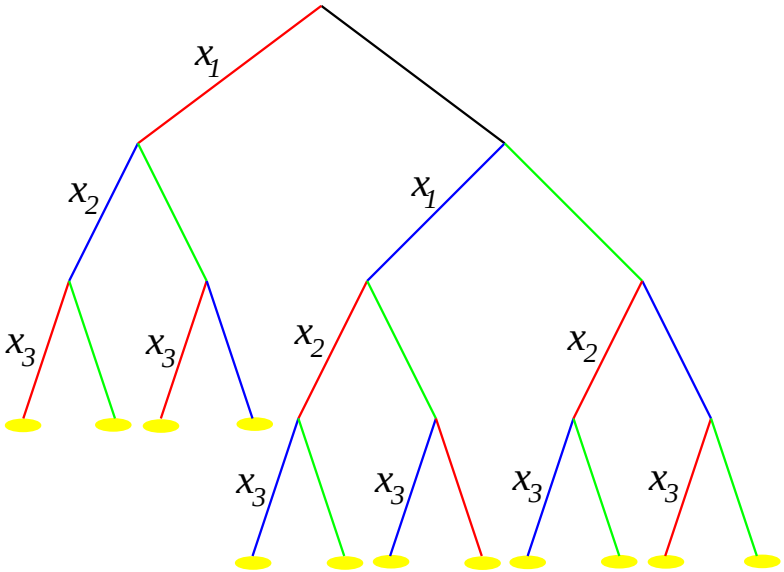
The order in which variables are chosen affects the search effort (to find a solution *or* to find all solutions)

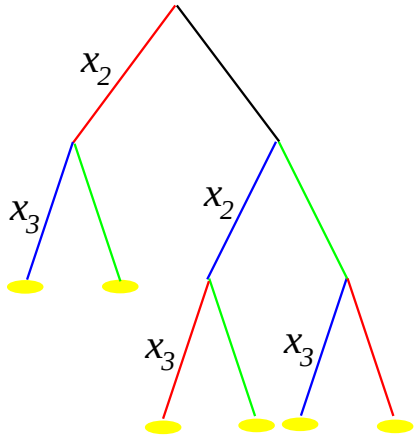


- Show that the nodes cannot be coloured with 3 colours (red,green,blue) so that adjacent nodes are different colours
- Assume that once values have been assigned to two of x_2, x_3, x_4, x_5 it can be detected that the other two nodes in the 4-clique cannot be coloured

Effect of Variable Ordering

If we assign x_1 first and then $x_2, x_3, \ldots$ we get a larger search tree than if we assign x_2, x_3 first:





Dynamic Variable Ordering

- It is not necessary to decide the order in which variables are assigned *before* search: the next variable can be chosen during the search, depending on what has happened so far
  - a common heuristic is to choose the variable with smallest remaining domain
  - (therefore likely to be the most constrained variable)
  - with this heuristic, after propagating the constraint $var \neq val$ on the right branch, the variable with smallest domain may no longer be var - so changing the current variable makes sense

Summary: Some Implications for Symmetry Breaking

Both constraint propagation and search are essential components of successfully solving CSPs:

- Adding constraints to the CSP to break symmetry can lead to domain reductions even before search starts
- Adding constraints to the right branch during search to break symmetry can also lead to additional constraint propagation
- The variable ordering can make a huge difference to the search effort
  - Any symmetry breaking method must work with, and not against, the chosen variable ordering
- Symmetry breaking should allow for different branching strategies e.g. domain splitting