Research Frontiers - Constraints: lecture 2

Karen Petrie
karenpetrie@computing.dundee.ac.uk

Formally

Variables and Values

- We have a set of variables, X1, X2, ..., Xn,
- all with domains D1, D2, ..., Dn
  - such that all variables Xi have a value in their respective domain Di.

Constraints

- There is also a set of constraints, C1, C2, ..., Cm,
  - such that a constraint Ci restricts (imposes a constraint on) the possible values in the domain of some subset of the variables.

Solutions

- A solution to a CSP is an assignment of every variable to some value in its domain such that every constraint is satisfied.
- Therefore, each assignment of a value to a variable must be consistent: it must not violate any of the constraints.

Finite Domains

- This course will only touch on problems that have finite domain variables.
- This means that the domains are a finite set of integers, as opposed to a real-valued domain that would include an infinite number of real-values between two bounds.

N-Queens

- Consider the popular N-Queens problem used throughout AI.
- This problem involves placing n queens on an n x n chessboard such that no queen is attacking another queen.

N-Queens (2)

- There are, of course, many ways to formulate this problem as a CSP
- (think: variables, domains, and constraints).

N-Queens Model

- A simple model is to represent each queen as a row so that
  - (for example) to solve the 4-queen problem, we have variables Q1, Q2, Q3, and Q4
- Each of these variables has an integer domain, whose values correspond to the different columns, 1-4

N-Queen Assignment

- An assignment consists of assigning a column to a queen,
  - i.e. { Q1 = 2 }, which "places" a queen in row 1, column 2

N-Queen Constraints

- The constraints on the problem restrict certain values for each variable so that all assignments are consistent.
- For example, after we have assigned Q1, and now want to assign Q2, we know we cannot use value 2, since this would violate a constraint.

N-Queen model in full

- Variables: { Q1, Q2, Q3, Q4 }
- Domain: { (1, 2, 3, 4), (1, 2, 3, 4), (1, 2, 3, 4), (1, 2, 3, 4) }
- Constraints: Alldifferent(Q1, Q2, Q3, Q4)
  and
  for i = 0...n and j = (i+1)...n, k = j-i, Q[i] != Q[j] + k and Q[i] != Q[j] - k.

Types of Constraints

Types of constraints

- If a constraint affects just a single variable, it is considered unary.
  - Unary constraints can be dealt with as a preprocessing step

Binary Constraints

- Constraints that involve two variables are binary constraints
  - they can be modelled as a constraint graph, where the nodes of the graph represent the variables and an edge connects two nodes if a constraint exists between the two variables.

Binary constraints contd

- A constraint of higher arity can always be reduced to a set of binary constraints!
  - (However, that doesn't mean that this is always a good idea--in some cases, the number of binary constraints for a problem can be exponential, thus creating an intractable model.)

Global Constraints

- More complex constraints, with arity > 2, are called global constraints.
- A simple example of a global constraint is the Alldifferent constraint; this constraint forces all the variables it touches to have different values.
  - (Note: It is easy to see how this particular global constraint could be decomposed into many "not equal" binary constraints.)

Search

- Searching a CSP involves first, choosing a variable, and then assigning a value to it.
- In a search tree, each node is a variable and branches leading away from that node are the different values that can be assigned to that variable.

Search

- Therefore, a CSP with n variables will generate a search tree of depth n.
- Successors are generated for only the current variable and the state of the problem is appended each time the search branches.

Search

- If we consider a simple depth-first search on a CSP, we realise that because of the constraints we have imposed, at some point during our search we may be unable to instantiate a variable because its domain is empty!
- In the case that we arrive at a node there are still unassigned variables and there are no branches leading away from that node, we must go backward.

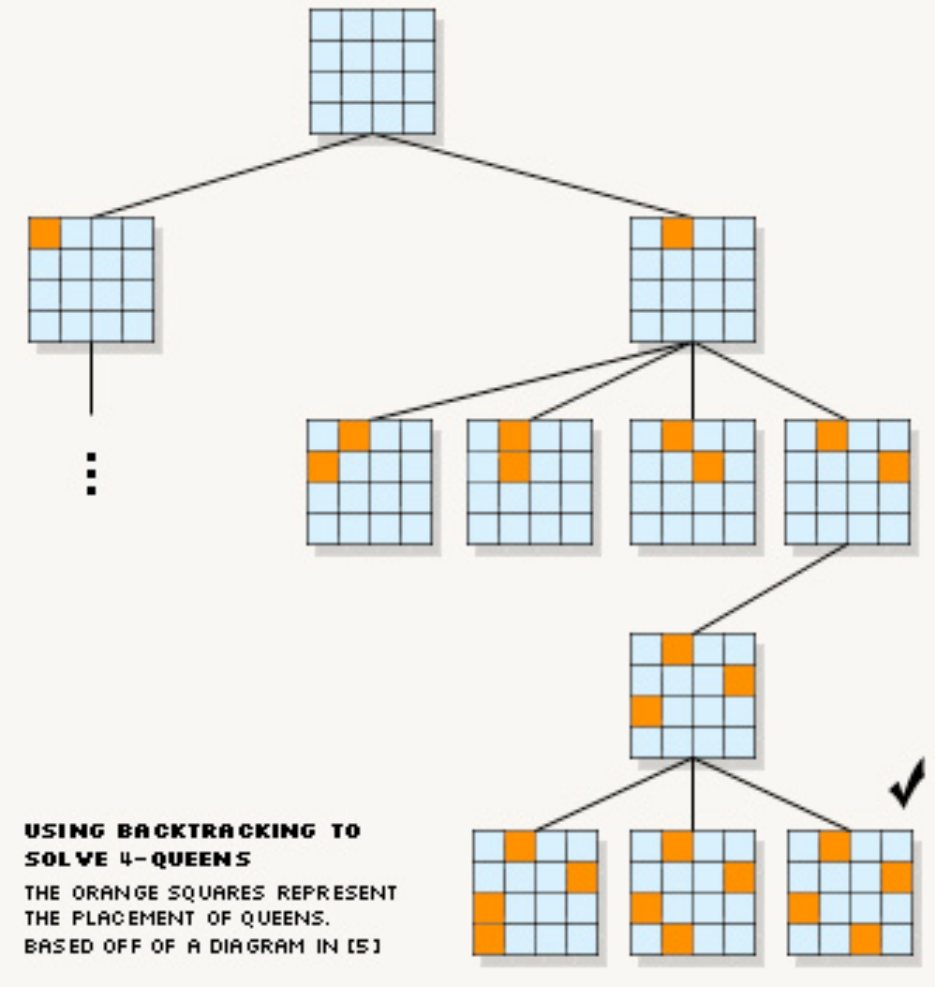
Main Algorithm (1)

- A variable is assigned a value and then the consistency of that assignment is checked.
- If the assignment is not consistent with the state of the problem, another value is assigned.

Main Algorithm (2)

- When a consistent value is found, another variable is chosen and this is repeated.
- If all values in the domain of a variable are inconsistent, the algorithm will backtrack to the previous assignment and assign it a new value.

Example


USING BACKTRACKING TO SOLVE 4-QUEENS
THE ORANGE SQUARES REPRESENT THE PLACEMENT OF QUEENS.
BASED OFF OF A DIAGRAM IN [5]

Constraint Propagation

- This has all been done just be checking assignments
- In reality we can be a bit more clever than that, by using propagation techniques.

Propagation

Problems with previous approach

- When our backtracking search chooses a value for a variable, it checks to see whether that assignment is consistent with the constraints on our problem.
  - Clearly, this is not very efficient.

Example

- Consider a simple graph-colouring problem:
- If there is an edge between two nodes, then once we assign a colour to one of these nodes, we know choosing that same colour for the other will not be consistent; therefore, we must temporarily remove the values from the domains of yet uninstantiated variables that are not consistent with the current problem state with the new assignment.

Forward Checking

- The forward checking algorithm does just this.
- Every time an assignment is made to a variable, all other variables connected to the variable (that is currently being instantiated) by constraints are retrieved and the values in their domains which are inconsistent with the current assignment are temporarily removed.

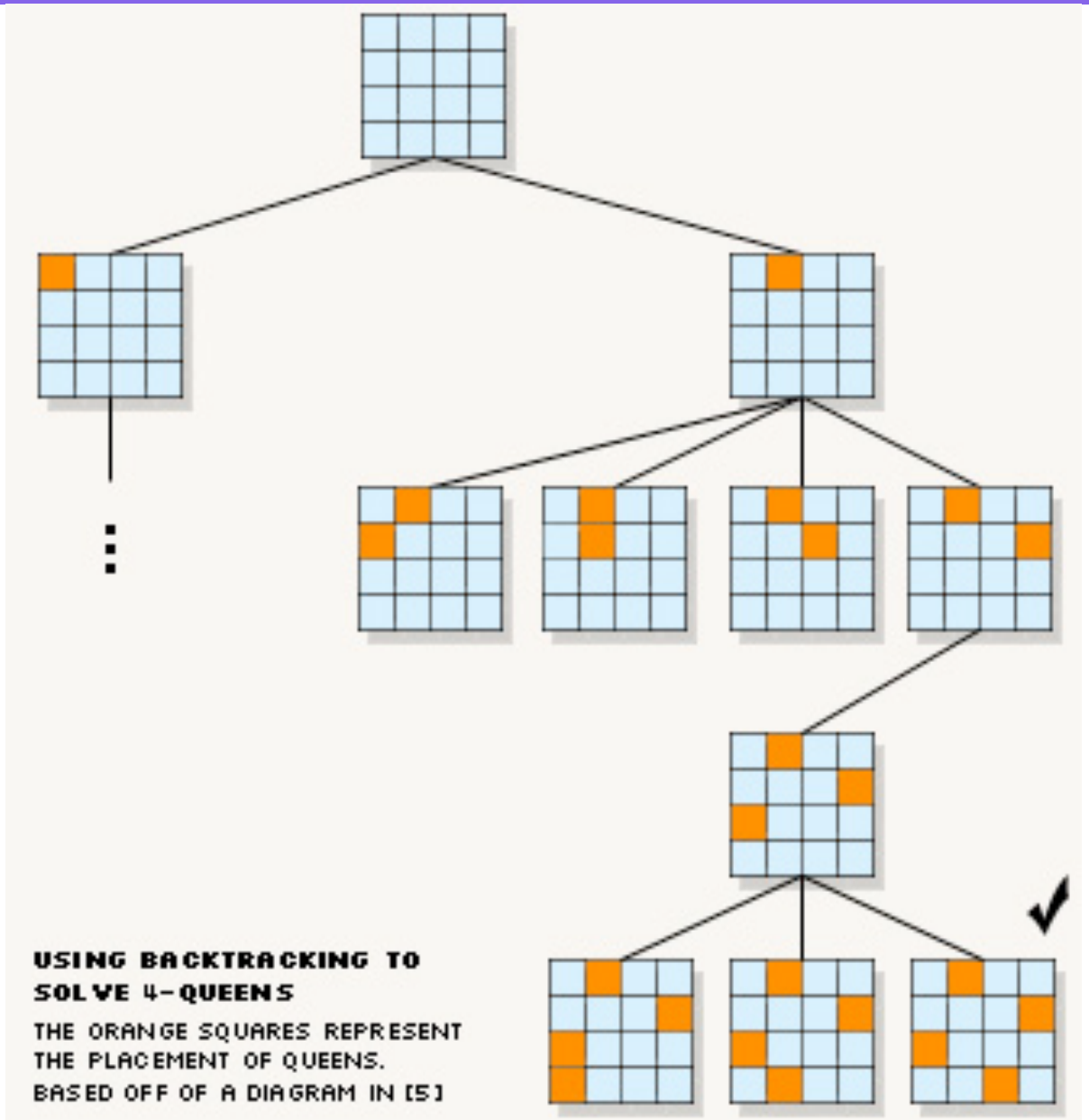
Forward checking

- In this way the domain of a variable can become empty and another value must be chosen for the current assignment.
- If there are no values left with which to assign the current variable, the search may need to backtrack, in which case those values that were temporarily removed as a result of the original assignment are reinstated to their respective domains.

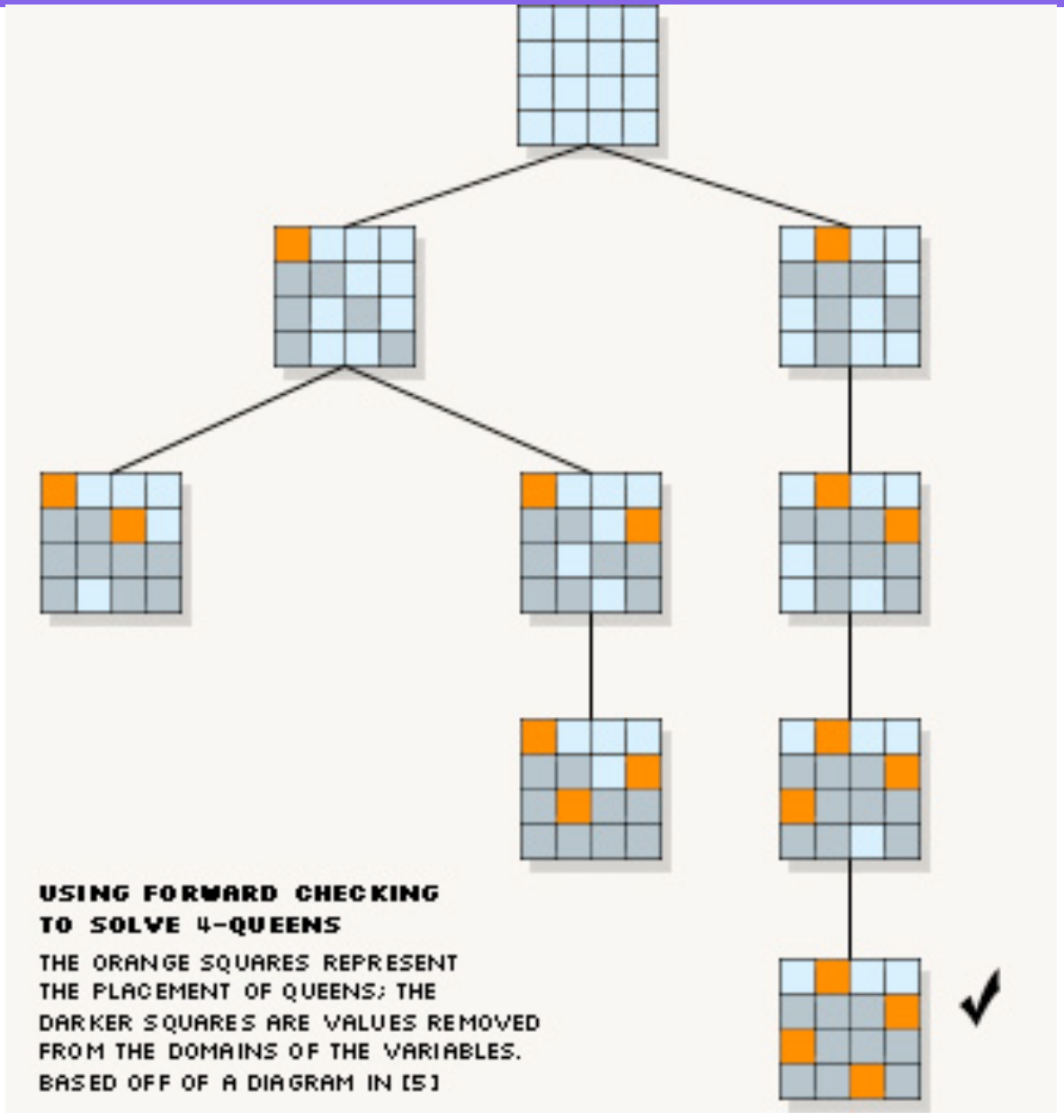
Forward Checking

- Forward checking is able to predict, in a sense, what assignments will lead to a failure and can act accordingly by pruning branches.
- It will, of course, encounter these inconsistencies much sooner than backtracking.

Backtracking Search Tree



Forward Checking Tree



USING FORWARD CHECKING TO SOLVE 4-QUEENS

THE ORANGE SQUARES REPRESENT THE PLACEMENT OF QUEENS; THE DARKER SQUARES ARE VALUES REMOVED FROM THE DOMAINS OF THE VARIABLES. BASED OFF OF A DIAGRAM IN [5]

Search: Local Consistency

- Forward checking utilizes a basic notion of consistency:
  - an assignment to a variable is consistent with other assignments given a set of constraints.
- k-consistency is a term that defines the extent to which constraints are propagated.

k-consistent

- By definition, a CSP is k-consistent if for any subset of $k > 1$ variables in the problem, a consistent assignment to one of those variables allows a consistent assignment to any kth variable.

Strongly k-consistent

- In addition to a problem being k-consistent, it can also be strongly k-consistent, which means it is consistent for k and all weaker consistencies less than k.

Benefit strongly k-consistent

- The benefit of a strongly k-consistent problem is that we will never have to backtrack!
- As with most things CSP, determining the correct level of consistency checking for a given problem is done empirically.

Node Consistency

- This is the weakest consistency check and simply assures that each variable is consistent with itself;
- if a variable is assigned a value, the value must be in that variables domain.

Arc Consistency

- 2-consistency is the most popular consistency and can be used as both a pre-processing step, or dynamically as part of the maintaining arc consistency (MAC) algorithm.

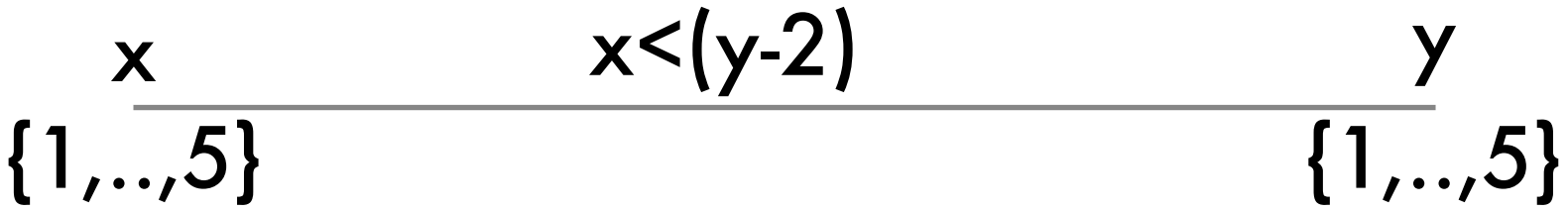
Arc Consistency

- The simple definition of arc consistency is that given a constraint C_{XY} between two variables X and Y, for any value of X, there is a consistent value that can be chosen for Y such that C_{XY} is satisfied, and visa versa.

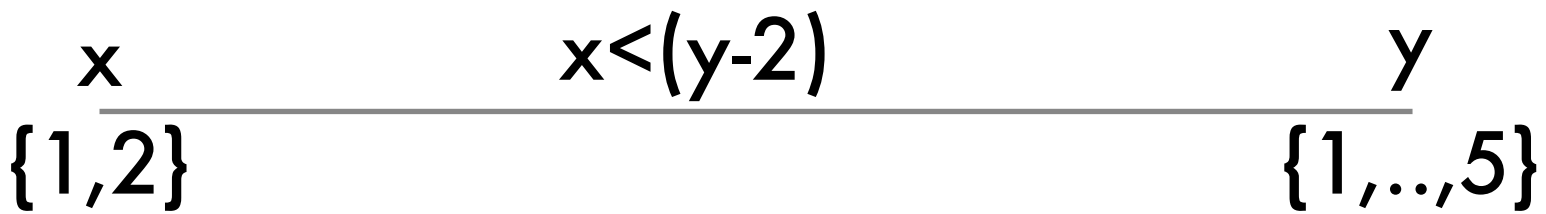
Arc Consistency

- Thus, unlike forward checking, arc consistency is directed and is checked in both directions for two connected variables.
- This makes it stronger than forward checking.

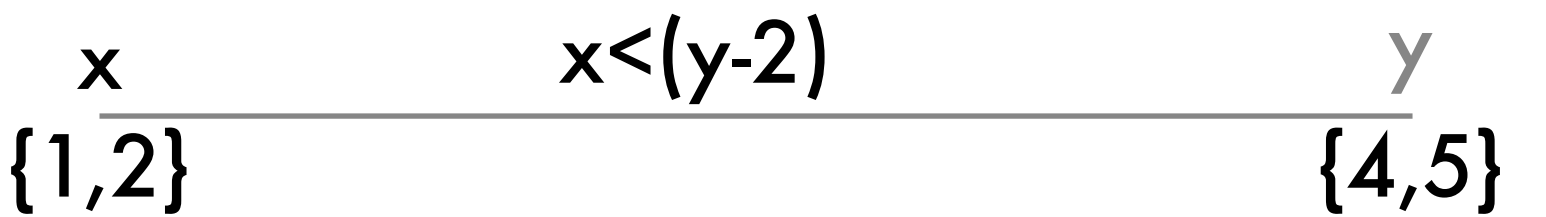
Arc Consistency Example



Original problem



(x,y) arc consistent



(x,y) and (y,x) arc consistent

Arc Consistency

- When applied as a preprocessing step, arc consistency removes all values from domains that are inconsistent with one another.
- If it is applied dynamically as MAC, the same algorithm that is used to check AC for pre-processing is applied after every variable instantiation during the search.

Path consistency

- The easiest way to think about path consistency is to consider a triangle, with three points labelled a, b, and c where edge(a, c) is not a solid line.
- This represents a problem where there are constraints between a and b, and b and c.
- Path consistency considers triples of variables, so that while a and c are not explicitly constrained, there is a constraint induced on them through the transitive nature of their constraints involving b.

Path Consistency

- It is of course possible to check every triple of variables in a binary CSP and tighten the constraints where possible.
- Clearly the number of possible triples is greater than the number of pairs of variables that need to be checked to make the problem arc consistent.
- This is one reason why path consistency algorithms are not in common use.

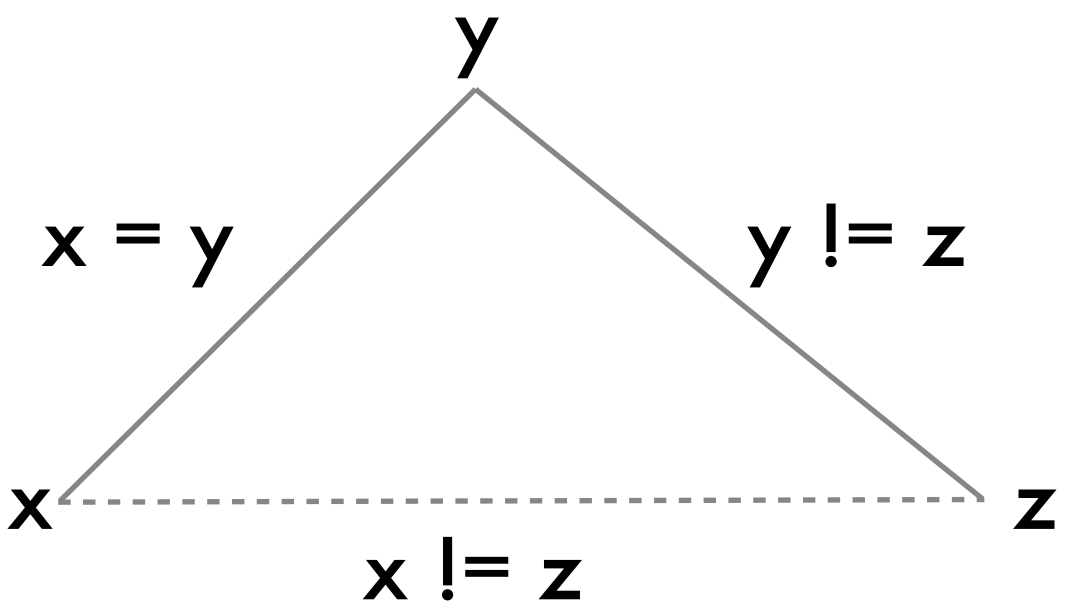
Path consistency (ctd)

- Another reason it is not used is:
  - since constraints are not generally expressed as allowed tuples of values it is not easy to remove individual pairs of values in order to tighten a binary constraint.

Path Consistency in practice

- In practice the constraints required for path consistency are often easy to see when the problem is formulated.
- See example:

Path Consistency example



Global constraints

Global Constraints

- The consistency levels we just discussed work well with binary constraints
- They do not work well with global constraints
- Remember that we could decompose into binary constraints but we do not want to.

Propagation for Global Constraints

- There are 2 different types of global propagation you should know:
  - GAC = Generalised Arc Consistent
  - BC = Bounds consistent

GAC

- GAC is where you make the most inference you can from that single constraint,
- deletes the maximum number of values from the domains of variables

Bounds Consistency

- Only operates on the lower bound or the upper bound of each variables domain;
- Tightens the domain

GAC vs BC

- So if you have the domain {1,..,5} of a variable
- GAC can change this to: {1,2,4}
- In the same case BC can change this is: {1,..,4}

Full CP Algorithm

Full Algorithm

- Now (finally) we have all the definitions we can consider how the full CP algorithm works.

Set-up

- The user, defines the variable values and constraints
- Internally the computer associates the bunch of variables associated with each constraint
- Any pre-processing takes place
  - Unary constraints are dealt with
  - May choose to do AC

Search

- From the variable and value ordering heuristic the first variable and value are chosen
- All the constraints associated with this variable are placed on the ‘constraint queue’

Propagation

- Each constraint in turn is propagated
  - made consistent to the level associated with the constraint
- If at this point any other variables are affected then the constraints associated with them are placed on the ‘constraint queue’
  - affected can mean ‘assigned’ or just that their domain is updated

Propagation (ctd)

- This process continues until there is nothing left on the constraint queue
  - The problem is consistent
- Or the domain of any variable becomes empty
  - domain wipeout

What next

- If the problem is consistent:
  - We continue by using the variable and value ordering heuristic to choose a new search variable
- If it is not consistent we backtrack

Algorithm Summary

```
Do setup;
Do any preprocessing;
while {solution not found} do
    Assign variable;
    while {constraint queue not empty} do
      make consistent;
    od;
    Check {domain wipeout}
    If domain wipeout =true then backtrack else
    continue;
 od;
```

Next Week

- No lecture
- Time to read Scheduling a Rehearsal by Barbara M. Smith
  - Available on MyDundee