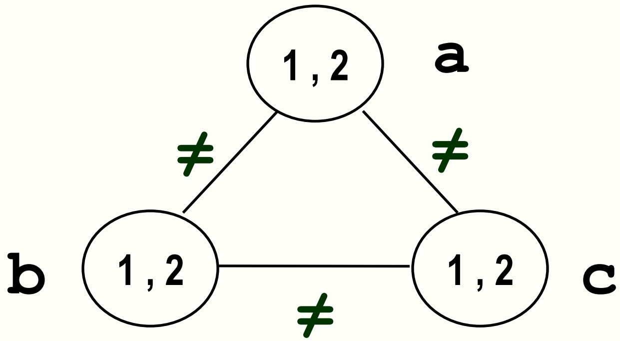
Constraint Programming

- An overview

- Higher Consistency Types: Path and i-consistency
- Consistency and Satisfiability
- Other Consistencies: Bounds- and SAC-Consistency
- Non-Binary Networks and Generalised Arc-Consistency

Path-Consistency

- The following constraint network is obviously inconsistent:



- Nevertheless, it is arc-consistent: every binary constraint of difference (≠) is arc-consistent whenever the constraint variables have at least 2 elements in their domains.

- However, is is not path-consistent: **no** label {<a-v_a>, <b-v_b>} that is consistent (i.e. does not violate any constraint) can be extended to the third variable (c).

  {<a-1>, <b-2>} $\rightarrow$ c ≠ 1, c ≠ 2 ; {<a-2>, <b-1>} $\rightarrow$ c ≠ 1, c ≠ 2

- This property is captured by the notion of path-consistency.

Path-Consistency

Definition (**Path Consistency**):

- A constraint satisfaction problem is path-consistent if,
  - It is arc-consistent; and
  - Every consistent 2-compound label $\{x_i\text{-}v_i,\ x_i\text{-}v_j\}$ can be extended to a consistent label with a third variable x_k ($k \neq i$ and $k \neq j$).

- The second condition is more easily understood as
  - For every compound label $\{x_i\text{-}v_i,\ x_j\text{-}v_j\}$, and for every k ($k \neq i$ and $k \neq j$) there must be a label $x_k\text{-}v_k$ that supports $\{x_i\text{-}v_i,\ x_j\text{-}v_j\}$. In other words, the compound label $\{x_i\text{-}v_i,\ x_j\text{-}v_j,\ x_j\text{-}v_{j,}\}$ satisfies constraints c_{ij}, c_{ik}, and c_{kj}.

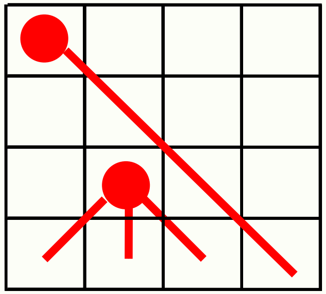
Path-Consistency

Example:

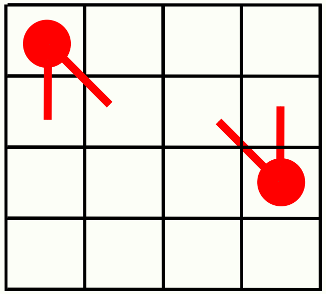
- By enforcing path consistency it is possible to avoid backtracking in the 4-queens problem.
- In fact, q_1-1 has only two supports in variable q_3, namely q_3-2 and q_3-4.

However:

- < q_1-1, q_3-2 > cannot be extended to variable q_4



- < q_1-1, q_3-4 > cannot be extended to variable q_2

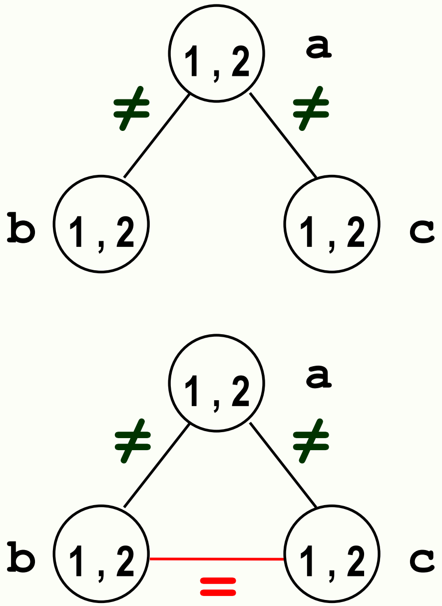


- Hence, value 1 can be safely removed from the domain of variable q_1.
- With similar reasoning, it may be shown that none of the corners, and none of the central positions can have a queen.

Path-Consistency

- In general, and despite the previous example, maintaining path consistency does not prune the domain of a variable, but rather prunes compound labels with cardinality 2.
- This means that, imposing arc-consistency on variables x_i and x_j through variable x_k, will tighten the (possible non-existing) constraint between x_i and x_j.

- In the example, a constraint of equality is imposed on variables **b** and **c**, because the compound labels { b-1 , c-2 } and { b-2 , c-1} cannot be extended to variable **a**.



Before path consistency

After path consistency

Path-Consistency

- The constraints that are imposed by maintaining arc-consistency can be very general, and are more easily understood if they are represented by means of Boolean matrices (i.e. by extension).

Example:

- Matrix $\mathbf{M_{ab}}$ encodes a binary constraint of difference(≠) between variables **a** and **b**, each with the same two values in their domains

M_{ab}	1	2
1	0	1
2	1	0

- Matrix M_{13} represents a **no_attack** constraint between queens in the 1st and 3rd rows, for the 4-queens problem.

M_{13}	1	2	3	4
1	0	1	0	1
2	1	0	1	0
3	0	1	0	1
4	1	0	1	0

Path-Consistency

- The imposition of path consistency, on variables x_i and x_j through variable x_k can be regarded as imposing a new constraint obtained by the Boolean multiplication of matrices M_{ik} and M_{jk}.

M_{14}	1	2	3	4
1	0	1	1	0
2	1	0	1	1
3	1	1	0	1
4	0	1	1	0

×

M_{43}	1	2	3	4
1	0	0	1	1
2	0	0	0	1
3	1	0	0	0
4	1	1	0	0

→

$C_{13\text{-}4}$	1	2	3	4
1	1	0	0	1
2	1	1	1	1
3	1	1	1	1
4	1	0	0	1

- The restriction to the initial constraint no_attack between queens 1 and 3, is imposed by **conjunction** of the initial matrix M_{13} with matrix $C_{13\text{-}4}$.

M_{13}	1	2	3	4
1	0	1	0	1
2	1	0	1	0
3	0	1	0	1
4	1	0	1	0

$\wedge$

$C_{13\text{-}4}$	1	2	3	4
1	1	0	0	1
2	1	1	1	1
3	1	1	1	1
4	1	0	0	1

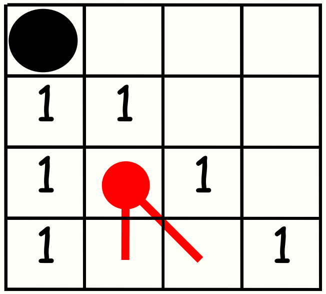
→

M_{13}	1	2	3	4
1	0	0	0	1
2	1	0	1	0
3	0	1	0	1
4	1	0	0	0

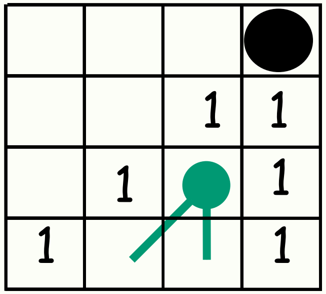
Path-Consistency

M'_{13}	1	2	3	4
1	0	0	0	1
2	1	0	1	1
3	1	1	0	1
4	1	0	0	0

- Indeed, the new matrix M'_{13} correctly registers the fact that



- Compound label $\{q_1\text{-}1, q_3\text{-}2\}$ does not have support on q_4 and is removed from the initial constraint c_{13}



- Compound label $\{q_1\text{-}4, q_3\text{-}3\}$ does not have support on q_4 and is removed from the initial constraint c_{13}

Path-Consistency

- The successive application of this tightening of the initial constraints will eventually lead to the deletion of values from the domains of the variables, as can be illustrated by the 4-queens problem.

- Firstly, constraint between variables q_1 and q_3 is tightened through variable q_2, as shown below.

1\2	1	2	3	4
1	0	0	1	1
2	0	0	0	1
3	1	0	0	0
4	1	1	0	0

2\3	1	2	3	4
1	0	0	1	1
2	0	0	0	1
3	1	0	0	0
4	1	1	0	0

1\3	1	2	3	4
1	0	1	0	1
2	1	0	1	0
3	0	1	0	1
4	1	0	1	0

1\3	1	2	3	4
1	0	1	0	0
2	1	0	0	0
3	0	0	0	1
4	0	0	1	0

- In this case, two compound labels $\{q_1\text{-}1, q_3\text{-}4\}$ and $\{q_1\text{-}4, q_3\text{-}1\}$ are removed from the initial constraint c_{13} (i.e. no_attack(q_1, q_3)).

- At this point, all values of both variables q_1 and q_3 still have supporting values in the domain of the other variable (non-null rows and columns)

Path-Consistency

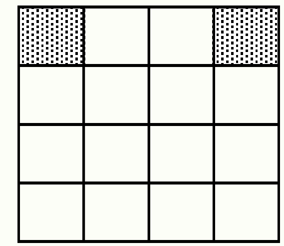
- Secondly, constraint c_{14} between variables q_1 and q_4 is tightened through variable q_3, as shown below.

1\3	1	2	3	4
1	0	1	0	0
2	1	0	0	0
3	0	0	0	1
4	0	0	1	0

3\4	1	2	3	4
1	0	0	1	1
2	0	0	0	1
3	1	0	0	0
4	1	1	0	0

1\4	1	2	3	4
1	0	1	1	0
2	1	0	1	1
3	1	1	0	1
4	0	1	1	0

1\4	1	2	3	4
1	0	0	0	0
2	0	0	1	1
3	1	1	0	0
4	0	0	0	0



- Notice
  - the use of the tightened constraint c_{13}.
  - The rows 1 and 4 are null in the new matrix M'_{13}.
- This last result means that values 1 and 4 from variable q_1 have no support on variable q_4 when the constraint c_{14} is tightened through variable q_3.
- Hence, values 1 and 4 can safely be removed from the domain of variable q_1

Path-Consistency

- The same applies when constraint c_{12}, is tightened through variable q_4.

- But first, the rows corresponding to values 1 and 4 of variable q_1 are set to zero, since these values were removed from the value of the variable.

1\4	1	2	3	4
1	0	1	1	0
2	0	0	1	1
3	1	1	0	0
4	0	1	1	0

→

1\4	1	2	3	4
1	0	0	0	0
2	0	0	1	1
3	1	1	0	0
4	0	0	0	0

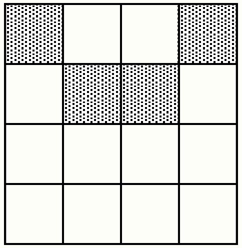
- The tightened constraint M_{14}, leads to the removal of values 2 and 3 from the domain of q_2, since both columns 2 and 3 become null.

1\4	1	2	3	4
1	0	0	0	0
2	0	0	1	1
3	1	1	0	0
4	0	0	0	0

4\2	1	2	3	4
1	0	1	0	1
2	1	0	1	0
3	0	1	0	1
4	1	0	1	0

1\2	1	2	3	4
1	0	0	1	1
2	0	0	0	1
3	1	0	0	0
4	1	1	0	0

1\2	1	2	3	4
1	0	0	0	0
2	0	0	0	1
3	1	0	0	0
4	0	0	0	0



Path-Consistency

- The process is repeated with the tightening of constraint c_{13}, through q_2.

- Since values 1 and 4 were removed from the domain of variable q_1 and so did values 2 and 3 from variable q_2, the corresponding rows and values are zero-es in matrix M_{12} as explained before.

  • And so do from matrix M_{23}.

1\2	1	2	3	4
1	0	0	1	1
2	0	0	0	1
3	1	0	0	0
4	1	1	0	0

→

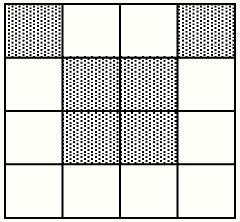
1\2	1	2	3	4
1	0	0	0	0
2	0	0	0	1
3	1	0	0	0
4	0	0	0	0

1\2	1	2	3	4
1	0	0	0	0
2	0	0	0	1
3	1	0	0	0
4	0	0	0	0

2\3	1	2	3	4
1	0	0	1	1
2	0	0	0	0
3	0	0	0	0
4	1	1	0	0

1\3	1	2	3	4
1	0	0	0	0
2	1	0	0	0
3	0	0	1	1
4	0	0	0	0

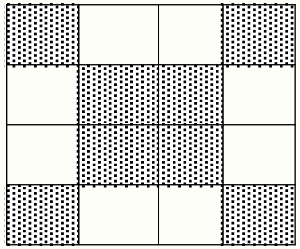
1\3	1	2	3	4
1	0	0	0	0
2	1	0	0	0
3	0	0	0	1
4	0	0	0	0



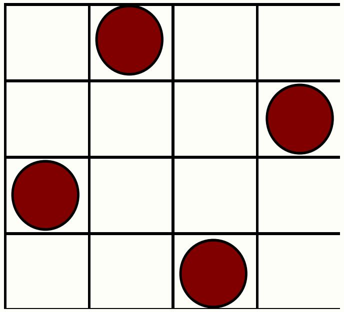
- Columns 2 and 3 are now made null the removal of these values from the domain of the corresponding variables in the new matrix, leading to the removal of 2 and 3 from the domain of q_3.

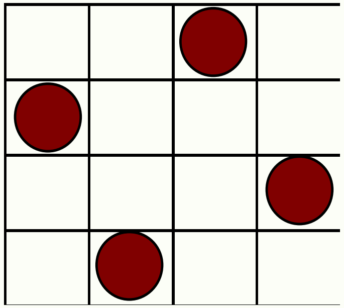
Path-Consistency

- Finally, the process is repeated with constraints involving variable q_4, leading to the removal of values 1 and 4 from the domain of q_4.



- At this point, the remaining values in variables q_1 to q_4 all belong to one of the two solutions of the 4-queens problem.



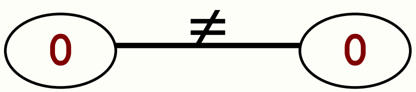


- Labelling of the variables can thus find a solution with no backtracking at all.

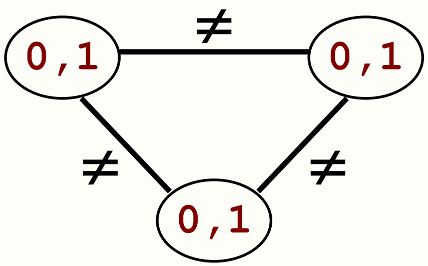
Binary Constraints: i-consistency

- The notions of node-, arc- and path-consistency can be generalised further to i-consistency, with increasing demands of consistency.

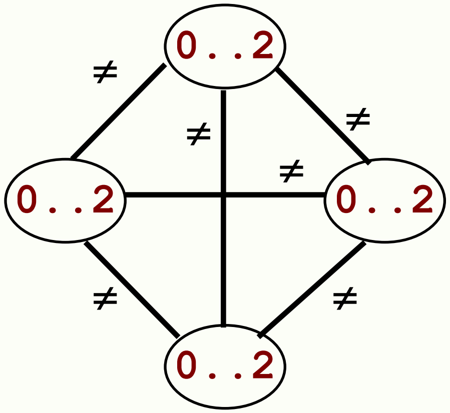
  - A node consistent network, that is not arc consistent (i.e. *2-consistent*).



  - An arc consistent network, that is not path consistent (i.e. *3-consistent*)



  - A path-consistent network, which is not …
    4-consistent



Binary Constraints: i-consistency

- The criterion of i-consistency is thus defined as follows.

Definition (i-Consistency):

- A constraint satisfaction problem is **i-consistent** if,
  - all compound labels of cardinality **i-1** can be extended to any other i^{th} variable.

- For example, any compound label < x_1-v_1, x_2-v_2, ..., x_k-v_k>, in a i-consistent constraint network (k = i-1), that satisfies the constraints over variables of the set S = $\{x_1, x_2, ..., x_k\}$ can be extended to any another variable x_i, ($\notin$ S) i.e. there is a value v_i in the domain of x_i that satisfies all the constraints defined on the set $S' = S \cup \{x_i\}$ of variables.
- As a special case, when i=1, only the unary constraints must be satisfied.

Binary Constraints: i-consistency

- Additionally, strong consistency can also be defined

Definition: i-Consistency

- A constraint satisfaction problem is **strongly i-consistent** if,
  - It is k-consistent for all k ≤ i.

- Given this definitions it is easy to notice the following equivalences:

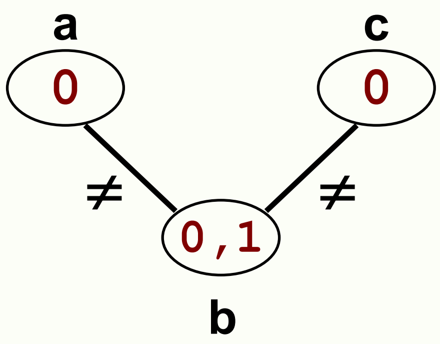
Node-consistency	↔	strong 1-consistency
Arc- consistency	↔	strong 2-consistency
Path-consistency	↔	strong 3-consistency

Binary Constraints: i-consistency

- Notice that the analogies of node-, arc- and path- consistency were made with respect to **strong i-consistency**.
- This is because a constraint network may be **i-consistent** but nonetheless not to be **m-consistent** (for some m < i).
  - For example, the network below is 3-consistent, but not 2-consistent. Hence it is not strongly 3-consistent.
    - The only 2-compound labels, that satisfy the constraints are

      {a-0,b-1}, {a-0,c-0}, and {b-1, c-0}
    - They may be extended to the remaining variable

      {a-0, b-1, c-0}
    - However, the 1-compound label {b-0} cannot be extended neither to variable a (i.e. {a-0,a-0} ?) nor c (i.e. {b-0, c-0} !



Binary Constraints: i-consistency

- For i > 3, i-consistency cannot be implemented with binary constraints alone. In fact:
  - 2-consistency checks whether a 1-label $\{x_i\text{-}v_i\}$ can be extended to some other 2-label $\{x_i\text{-}v_i,\ x_j\text{-}v_j\}$. If that is not the case, label $\{x_i\text{-}v_i\}$ is removed from the domain of X_i.
  - 3-consistency checks whether a 2-label $\{x_i\text{-}v_i,\ x_j\text{-}v_j\}$ can be extended to a 3-label $\{x_i\text{-}v_i,\ x_j\text{-}v_j,\ x_k\text{-}v_k\}$. If that is not the case, label $\{x_i\text{-}v_i,\ x_j\text{-}v_j\}$ is removed.
  - Removing label $\{x_i\text{-}v_i,\ x_j\text{-}v_j\}$ is not achieved by removing values from the domains of the variables, but rather by tightening a constraint c_{ij} on variables x_i and x_j.
- By analogy, to impose 4-consistency 3-labels may have to be removed, hence a constraint on 3 variables has to be created or tightened.
- In general, maintaining i-consistency requires imposing constraints of arity i-1.

Binary Constraints: i-consistency

- The algorithms that were presented for achieving arc-consistency could be adapted to obtain i-consistency, provided that we consider constraints with arity i-1.
- The adaptation of the AC-1 algorithm (brute-force) would have
  - Space complexity of $\mathbf{O(2^i (nd)^{2i})}$.
  - Time complexity of $\mathbf{O(n^i d^i)}$.
- The adaptation of the AC-4 and AC-6 algorithms lead to optimal asymptotic time complexity of Ω $\mathbf{(n^i d^i)}$ (a lower bound).
- Given the mentioned complexity (even if the typical cases are not so bad) their use in backtrack search is generally not considered.
- The main application of these criteria is in cases where tractability can be proved based on these criteria.

Network Consistency and Satisfiability

- All types of i-consistency can be imposed by polynomial algorithms, with asymptotic time complexity $\Omega(n^i d^i)$ even when the corresponding problems (modelled with binary constraints) are NP-complete.
- Hence, in general for a network with **n** variables, i-consistency (for any i < n) does not imply satisfiability of the problem, i.e.
  - There are unsatisfiable problems (modelled with binary constraints) whose corresponding network is i-consistent.
- Of course, the converse is also true
  - There are satisfiable problems, modelled with binary constraints, whose corresponding network is not i-consistent.
- Nevertheless, in some special cases, the two concepts (i-consistency and satisfiability are equivalent).
- We will overview three such cases.

Network Consistency and Satisfiability

Case 1: A network of binary constraints, whose variables have only 2 values in their domain, is satisfiable iff it can be made path-consistent.

Proof: By recasting the problem to 2-SAT.

- If the network is path-consistent, then
  1. all binary constraints are explicit, and
  2. the matrices representing the constraints have a maximum of 2 rows and 2 columns.
- Hence, the satisfaction of a constraint can be equated to the satisfaction of a Boolean formula in disjunctive normal form (see figure below for an example).

a\b	3	4
2	1	1
5	0	1

$$(a2 \wedge b3) \vee (a2 \wedge b4) \vee (a5 \wedge b4)$$

Network Consistency and Satisfiability

- But given that there are only two values in each domain we may make explicit that one of the values corresponds to the negation of the other, as shown below

a\b	3	4
2	1	1
5	0	1

a2 = a

a5 = ¬a

b3 = b

b4 = ¬b

$$R = (a \wedge b) \vee (a \wedge \neg b) \vee (\neg a \wedge \neg b)$$

- Now, since path-consistency makes explicit all implicit relations between variables, the corresponding path-consistent network will contain a 0-matrix if and only if the corresponding problem is unsatisfiable.

- Since all the constraints are recast into a 2-SAT formula, solving them is tractable!

Graph Width

- Before presenting another theorem relating k-consistency and tractability it is convenient to consider constraint networks with n-ary constraints (n>2), either because a problem is specified with such constraints, or because these constraints are induced in a (binary) graph when k-consistency (k>3) is imposed on the constraint network.
- For this purpose we have the following definition:

Definition: Primal Graph of a Constraint Network

- The primal graph of a constraint network is a graph where there is an edge between two variables iff there is some constraint with the two variables in its scope.

- Given the definition, the primal graph of a constraint satisfaction problem coincides with the problem graph if the only constraints to be considered are binary (or unary).

Graph Width

Example:

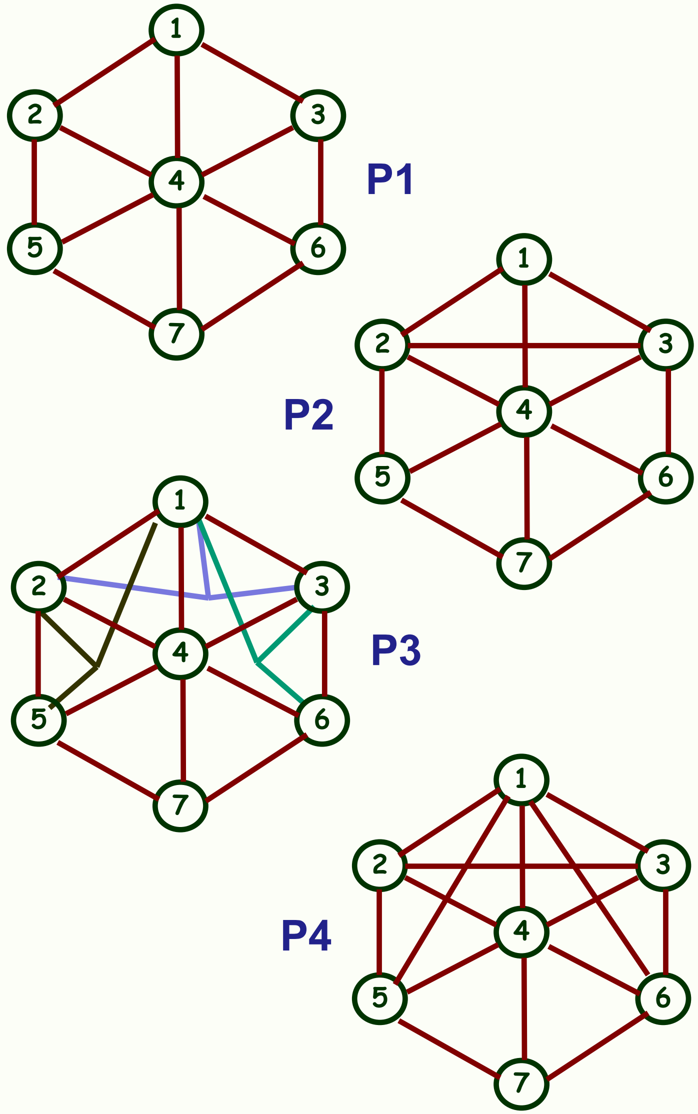
- Let us assume that:

1. the initial formalisation of a problem leads to the network P1;

2. imposing path-consistency, arcs are added between variables, e.g. 2-3, resulting in network P2 (still a graph);

3. Imposing 4-consistency, hyper-arcs are imposed on variables 1-2-3, 1-2-5 and 1-3-6, resulting in network P3 (a hyper-graph);

- Now, the primal graph of the problem is shown as graph P4.



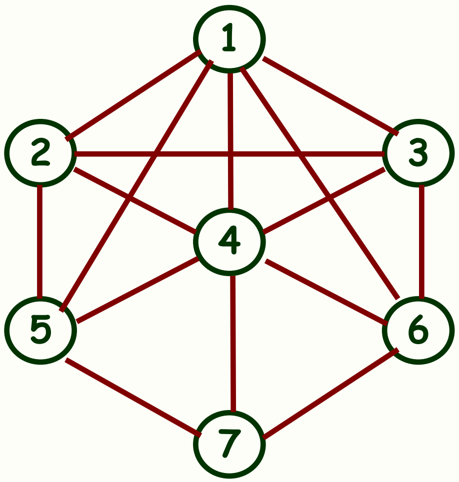
Graph Width

Definition: Node width, given ordering O

- Given some total ordering, **O**, defined on the nodes of a graph, the width of a node **N** given ordering **O,** is the number of lower order nodes that are adjacent to **N**.

Example: For the graph and the ordering O_1 shown we have

- $w(1, O_1) = 0$
- $w(2, O_1) = 1$ (node 1)
- $w(3, O_1) = 2$ (nodes 1 and 2)
- $w(4, O_1) = 3$ (nodes 1, 2 and 3)
- $w(5, O_1) = 3$ (nodes 1, 2 and 4)
- $w(6, O_1) = 3$ (nodes 1, 3 and 4)
- $w(7, O_1) = 3$ (nodes 4, 5 and 6)

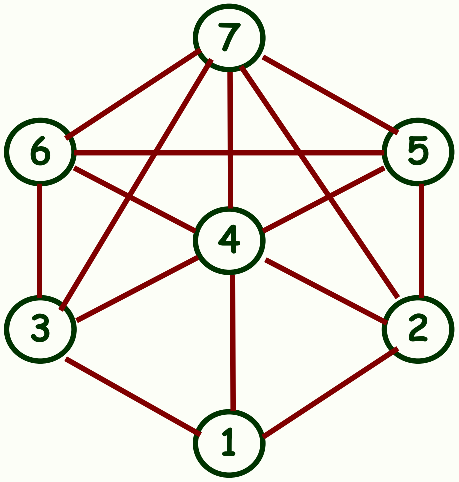


Graph Width

- Different orderings will produce different widths for the nodes of the graphs.

Example: For the same graph but with an “inverted ordering O_2, we have

- $w(1, O_2) = 0$
- $w(2, O_2) = 1$ (node 1)
- $w(3, O_2) = 1$ (node 1)
- $w(4, O_2) = 3$ (nodes 1, 2 and 3)
- $w(5, O_2) = 2$ (nodes 2 and 4)
- $w(6, O_2) = 2$ (nodes 3 and 4)
- $w(7, O_2) = 5$ (nodes 2, 3, 4, 5 and 6)



Graph Width

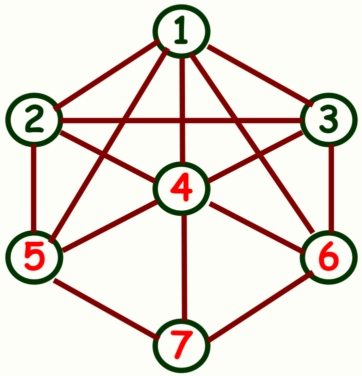
- From the width of the nodes one may obtain the width of a graph.

Definition: Graph width, given ordering O

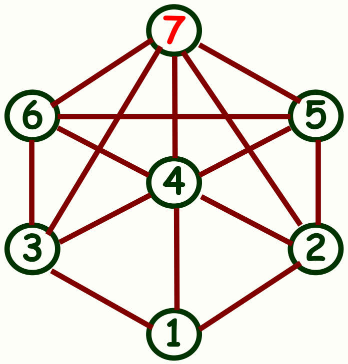
Given some total ordering, O, defined on the nodes of a graph, the width of the graph, given ordering O is the maximum width of its nodes, given ordering O.

Example: For the two orderings we obtain

$W(G,O_1) = 3$



$W(G,O_2) = 5$



Graph Width

- Now we may define the width of a graph, independent of the ordering used.

Definition: Graph width

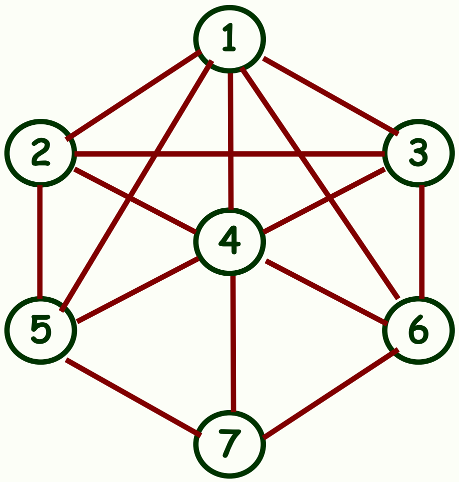
- The width of a graph is the lowest width of the graph over all possible total orderings.

- In the example, it is easy to see that the width of the graph is 3.

a) Ordering O_1 assigns width 3 to the graph. Hence the graph width is not greater than 3.

b) A width of 2 on a graph with 7 nodes would require the graph to have at most 0+1+5*2 = 11 edges. Hence, the width of the graph, which has 15 edges, cannot be less than 3.

c) From a) and b) the width of graph G is 3.



Tractability and i-Consistency

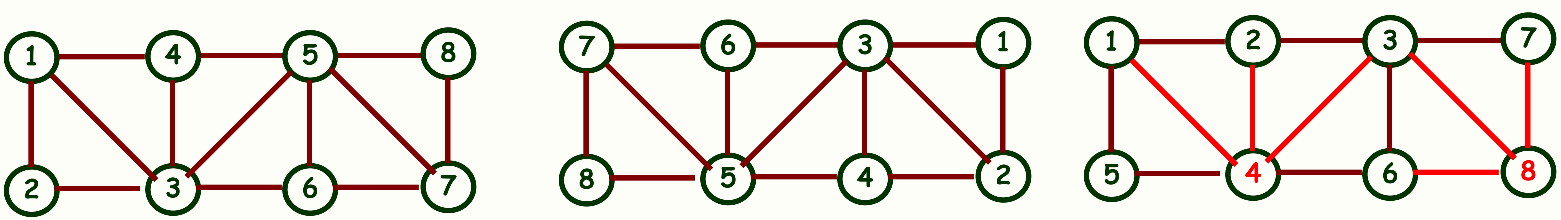
- Now we can present a theorem relating k-consistency and the width of a graph, which indirectly checks whether a problem is tractable.

Theorem: Graph width and Satisfiability

- Let a constraint satisfaction problem be modelled by a constraint network, that after imposing k-consistency leads to a primal graph of width k-1. Under these conditions, any ordering that assigns width k to the primal graph is a backtrack free ordering (BTF).

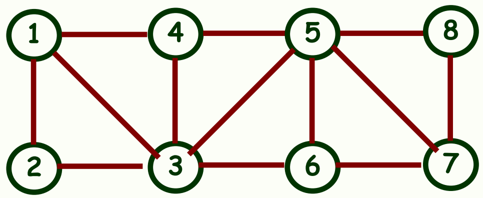
Example:

- For the networks bellow, assumed to be path-consistent (strong 3-consistent) O_1 and O_2 (with widths 2) are BTF orderings, but O_3 (width 3) is not.



Tractability and i-Consistency

- In fact, for ordering O_1

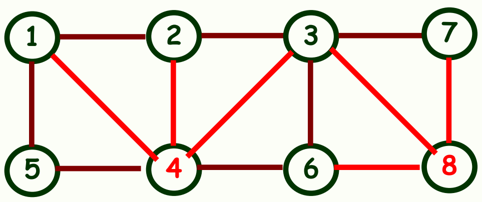


1. every label $\{x_1\text{-}v_1,\ x_2\text{-}v_2\}$, has a support in x_3, say $\{x_3\text{-}v_3\}$.
2. But, label $\{x_1\text{-}v_1,\ x_3\text{-}v_3\}$, has a support in x_4, say $\{x_4\text{-}v_4\}$.
3. Now, label $\{x_3\text{-}v_3,\ x_4\text{-}v_4\}$, has a support in x_5, say $\{x_5\text{-}v_5\}$.
4. Then, label $\{x_3\text{-}v_3,\ x_5\text{-}v_5\}$, has a support in x_6, say $\{x_6\text{-}v_6\}$.
5. And, label $\{x_5\text{-}v_5,\ x_6\text{-}v_6\}$, has a support in x_7, say $\{x_7\text{-}v_7\}$.
6. Finally, label $\{x_5\text{-}v_5,\ x_7\text{-}v_7\}$, has a support in x_8, say $\{x_8\text{-}v_8\}$.

- All things considered, label $\{x_1\text{-}v_1,\ x_2\text{-}v_2,\ x_3\text{-}v_3,\ x_4\text{-}v_4,\ x_5\text{-}v_5,\ x_6\text{-}v_6,\ x_7\text{-}v_7, x_8\text{-}v_8\}$ is a solution of the problem, and was found with no backtracking

Tractability and i-Consistency

- However, for ordering O_3
  - every label $\{x_1\text{-}v_1, x_2\text{-}v_2\}$, has a support in x_4, say $\{x_4\text{-}u_4\}$.
  - every label $\{x_2\text{-}v_2, x_3\text{-}v_3\}$, has a support in x_4, say $\{x_4\text{-}v_4\}$.



- But there is no guarantee that v_4 and u_4 are the same!
- In fact, there might be no value in the domain of x_4 that supports both the assignments $\{x_1\text{-}v_1, x_2\text{-}v_2\}$, and $\{x_2\text{-}v_2, x_3\text{-}v_3\}$.
- If this is the case, after assigning values $\{x_1\text{-}v_1, x_2\text{-}v_2, x_3\text{-}v_3\}$, no value exists for x_4 that is compatible with these and one of them must be backtracked !!!
- In this example, the same would happen with variable x_8 (connected to "prior" variables x_3, x_6 and x_7).

Graph Width

- To take advantage of the relation between i-consistency and induced graph width, it is still necessary to find the width of a graph or, equivalently, one optimal ordering, i.e. one that induces a minimal width.

- Fortunately there is a greedy algorithm (thus polynomial) that finds all optimal orderings. The idea is very simple. Always select (non-deterministically) a node with the least number of adjacent nodes (less degree). Put it in the back of the ordering, delete all the arcs leading to the node, and proceed recursively.

```
function min-width(G: set of Nodes, A: set of Arcs):
         Sequence of Nodes;
   if G.nodes = {n} then
      L ← [n]
   else
      n <- argn min {degree(n,G,A)}
      G1.arcs ← G.arcs \ {A: A = (_,n)  ∨ A = (n,_)
      G1.nodes ← G.nodes\{N}
      L ← min-width(G1) + [ n ]
   end if
   min-width ← L
 end function
```

Network Consistency and Satisfiability

- So, in addition to

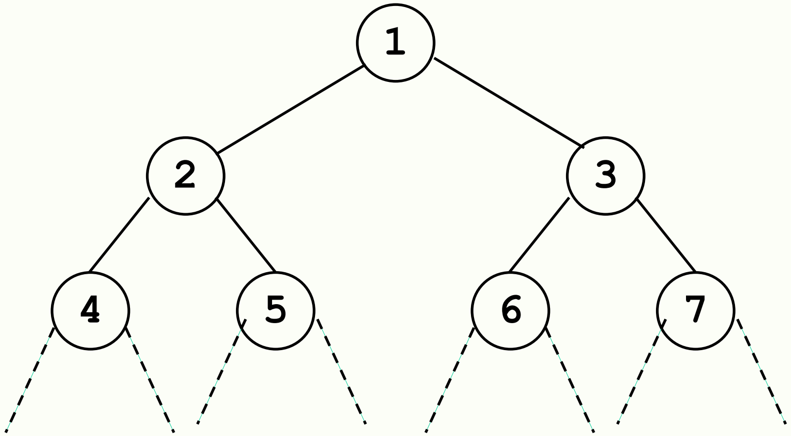
Case 1: A network of binary constraints, whose variables have only 2 values in their domain, is satisfiable iff it can be made path-consistent.

we have

Case 2: A network of constraints (of any arity), whose primal graph has width k is satisfiable iff it is k+1-consistent.

Example:

- 2-consistency (i.e. arc-consistency) of the constraint network guarantees the satisfaction of the associated constraint problem, if all constraints are binary and the constraint graph has the topology of a tree.
- A BTF ordering proceeds from the root to the leaves



Network Consistency and Satisfiability

- The previous 2 cases can be regarded as special cases of CSP tractable problems whose **language** or **structure** are restricted wrt to general binary CSPs.

Case 1 (Constraint Language Restriction): A network of binary constraints, whose variables have only 2 values in their domain, is satisfiable iff it can be made path-consistent.

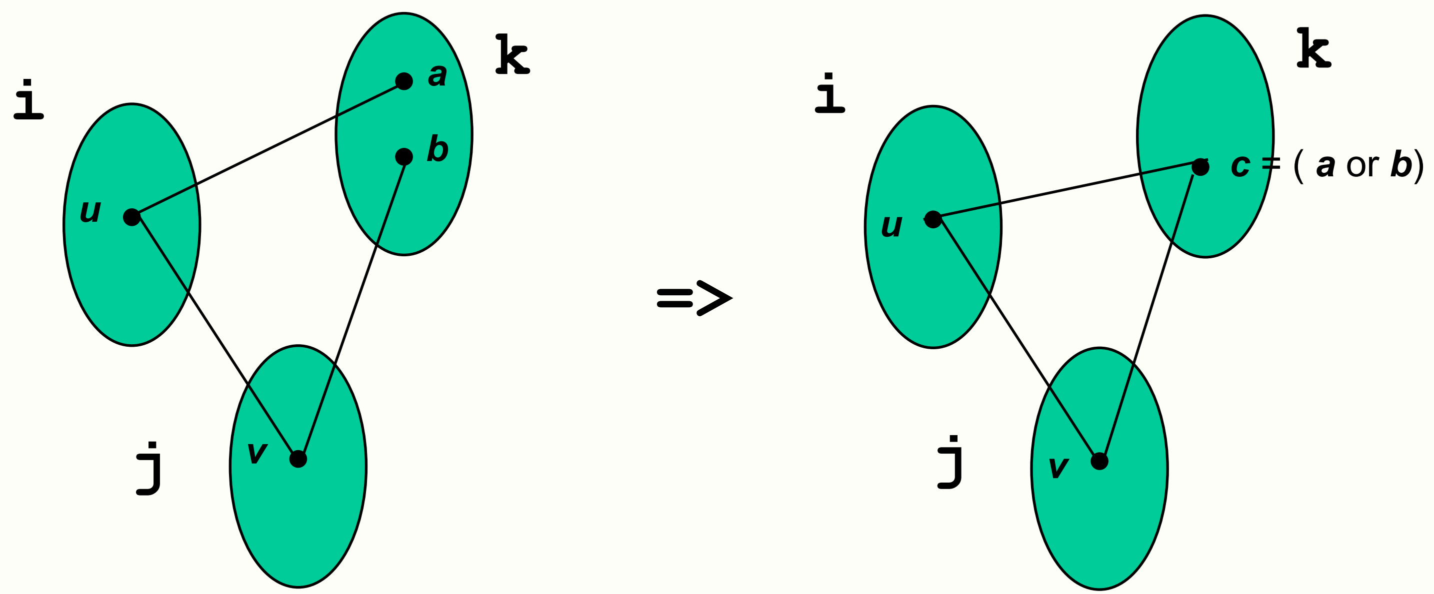
Case 2 (Structural Restriction): A network of constraints (of any arity), whose primal graph has width k, is satisfiable iff it is k+1-consistent.

- For the third case we present next, the **Broken-Triangle Property** (BTP) is a polynomial-time detectable property which defines a novel hybrid tractable class of binary CSP instances.
- The BTP can be viewed as forbidding the occurrence of certain sub-problems of a fixed size within a CSP instance.

Network Consistency and Satisfiability

Definition: Broken-triangle property

- A binary CSP instance satisfies the broken-triangle property (BTP) with respect to the variable ordering <, if, for all triples of variables i, j, k such that i < j < k, if $(u,v)\in R_{ij}$, $(u,a)\in R_{ik}$ and $(v,b)\in R_{jk}$, then either $(u,b)\in R_{ik}$ or $(v,a) \in R_{jk}$.



Network Consistency and Satisfiability

- To check the tractability of this class of problems we have the following *

Theorem: Detection of a BTP variable ordering

- Given a binary CSP instance I, there is a polynomial-time algorithm to find a variable ordering <, such that I satisfies the broken-triangle property with respect to < (or to determine that no such ordering exists).

- For the CSP instances that have the BTP with respect to some ordering there is thus a polynomial-time procedure to determine a variable ordering which guarantees backtrack-free search.

- Moreover,

Network Consistency and Satisfiability

Theorem: Finding solution for a BTP instance

- For any binary CSP instance which satisfies the BTP with respect to some known variable ordering <, it is possible to find a solution in $O(d^2e)$ time (or determine that no solution exists).

- Hence a problem that presents the BTP property is tractable.
  - Not only it is tractable finding the order of variables; but
  - finding a solution in BTP orderings is also tractable.

* See details in Martin C. Cooper, Peter G. Jeavons, András Z. Salamon, Generalizing constraint satisfaction on trees: Hybrid tractability and variable elimination, AI Journal, 174 (2010), pp. 570–584

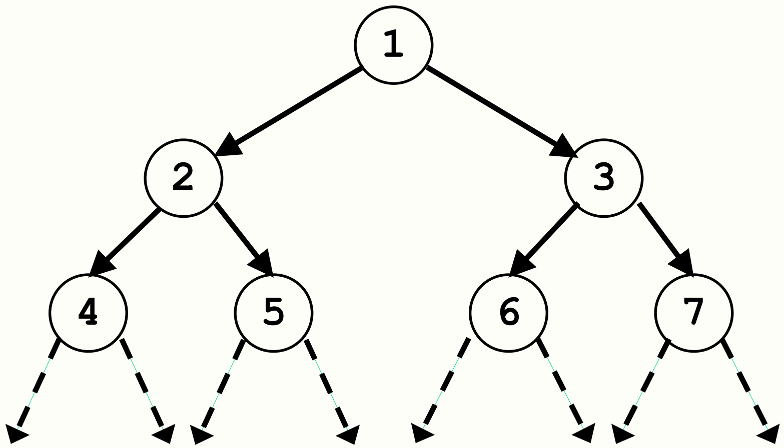
Directed Arc-consistency

- Some constraints may take advantage of some special features to improve the efficiency of their (arc-consistency) propagators.

- Take for example the case of a CSP with a tree-structure.

- Although arc-consistency requires support in both directions of the edges of the graph, support is only needed “downwards”, if the the order in which variables are labelled is also “downwards”.

- Hence, in these networks there is only the need to maintain **directed-arc consistency!**

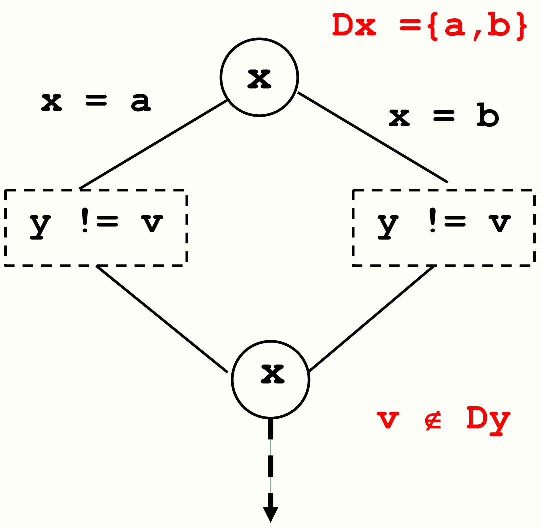


- Of course, this case can be generalised for networks of width k for which all that is required is to maintain **directed k-consistency** to guarantee satisfiability.

Singleton Arc-consistency

- As mentioned, path-consistency is usually too heavy. Nevertheless, there is a variation of arc-consistency that is sometimes able to prune values from variables that standard arc-consistency cannot. An example can illustrate this effect.

- If at some point in the search, some variable x is chosen to be labelled, one may try to label it with all its possible values, and apply arc-consistency with no commitment (sometimes known as “shaving”).



- If some value v of some other unlabelled variable y is removed in all cases, than

- this value can safely be removed form the domain of y, below the choice-point where variable x is labelled.

Arc-consistency: special purpose propagators

- Some constraints may take advantage of some special features to improve the efficiency of their propagators.
- Take for example the propagator for the n-queens problem: **no_attack(i, q_i, j, q_j)**.
- The usual arc-consistency would propagate the constraint (i.e. prune each of the values in the domain of q_1/q_2 with no supporting value in q_2/q_1), whenever the constraint is taken from the queue (assuming an AC-3 type algorithm).
- However, it is easy to see that a queen with 4 values in the domain offers at least one support value to any other queen.
- In fact a queen q_i can only be attacked by 3 positions of a queen q_j from another row j. Hence the 4^{th} queen in row j will not attack it.
- Hence, the propagator for **no_attack** should first check the cardinality of the domains, and only check for supports for queens that have a domain with cardinality of 3 or less!

Non-Binary Constraints: Bounds-consistency

- In numerical constraints (equality and inequality constraints) it is very usual not to impose a too demanding arc-consistency, but rather to impose mere **bounds consistency**.
- Take for example the simple constraint **`a < b`** over variables **`a`** and **`b`** with domains 0..1000.
- In such inequality constraints, the only values worth considering for removal are related to the bounds of the domains of these variables.
- In particular, the above constraint can be compiled into

  `max(a) < max(b)` and **`min(b) < min(a)`**

- In practice this means that the values that can be safely removed are
  - all values of **a** above the maximum value of **b**;
  - all values of **b** below the minimum value of **a**;
- These values can be easily removed from the domains of the variables.

Non-Binary Constraints: Bounds-consistency

- It is interesting to note how this kind of consistency detects contradictions.
- Take the example of **a < b** and **b < a**, two clearly unsatisfiable constraints. If the domains of **a** and **b** are the range **1..1000**, it will take about **500** iterations to detect contradiction

a:: 1 .. 1000, b:: 1 .. 1000	**a < b →**	**a:: 1 .. 999, b:: 2 .. 1000**
a:: 1 .. 999, b:: 2 .. 1000	**a > b →**	**a:: 3 .. 999, b:: 2 .. 998**
a:: 3 .. 999, b:: 2 .. 998	**a < b →**	**a:: 3 .. 997, b:: 4 .. 998**
a:: 3 .. 997, b:: 4 .. 998	**a > b →**	**a:: 5 .. 997, b:: 4 .. 996**
	....	
a:: 499..501, b:: 498..500	**a < b →**	**a::499..499, b::500..500**
a:: 500..500, b:: 500..500	**a > b →**	**a::501..500, b::500..499**

- Now, the lower bound is greater than the upper bound of the variables domains, which indicates contradiction!

Non-Binary Constraints: Bounds-consistency

- This reasoning can be extended to more complex numerical constraints involving numerical expressions:

Example:

$$a + b \leq c$$

- The usual compilation of this constraint is

$\max(a) \leq \max(c) - \min(b)$	to prune high values of **a**
$\max(b) \leq \max(c) - \min(a)$	to prune high values of **b**
$\min(c) \geq \min(a) + \min(b)$	to prune high values of **a**

- Many numerical relations envolving more than two variables can be compiled this way, so that the corresponding propagators achieve bounds consistency.

- This is particularly useful when the domains are encoded not as lists of elements but as pairs **min .. max** as is usually the case for numerical variables.

Enforcing generalised arc-consistency: GAC-3

- All algorithms for achieving arc-consistency can be adapted to achieve **generalised arc-consistency** (or **domain-consistency**) by using a modified version of the revise_dom predicate, that for every k-ary constraint checks support values from each variable in the remaining k-1 variables.

```
predicate revise_gac(V,D, c ∈ C): set of labels;
   R <- ∅;
   for xi in vars(c)
      for vi in dom(Xi) do
      Y  = vars(c) \ {xi} ;
      if ¬ ∃ V in dom(Y): satisfies({xi-vi, Y-V}, c) then
         dom(Xi) <- dom(xi) \ {vi};
         R <- R ∪ {xi-i};
      end if
   end for
   revise_gac <- R;
end predicate
```

Enforcing generalised arc-consistency: GAC-3

- The GAC-3 algorithm is presented below, as an adaptation of AC-3.
- Any time a value is removed from a variable X_i, all constraints that have this variable in the scope are placed back in the queue for assessing their local consistency.

```
procedure AC-3(V, D, C);
   NC-1(V,D,C);        % node consistency
   Q = { c | c ∈ C};
   while Q ≠ ∅ do
      Q = Q \ {c}   % removes an element from Q
      for xi-i in revise_gac(V,D, c ∈ C) do   % revised xi
         Q = Q ∪ {r | r ∈ C ∧  i ∈ vars(r) ∧ r ≠ c }
      end if
   end while
end procedure
```

Complexity of GAC-3

Time Complexity of **GAC-3: $O(a\ k^2\ d^{k+1})$**

- Every time that an hyper-arc/n-ary constraint is removed from the queue Q, predicate revise_gac is called, to check at most $k*d^k$ tuples of values.
- In the worst case, each of the **a** constraints is placed into the queue at most **k*d** times.
- All things considered, the worst case time complexity of GAC-3, is

$$O(a\ k^2\ d^{k+1})$$

- Of course, when all the constraint are binary the complexity of GAC-3 is the same of AC-3, i.e.

$$O(a\ d^3)$$

Constraint Propagation

- Generalised arc-consistency provides a scheme for an architecture of constraint solvers, even when constraints are not binary.
- For every constraint (binary or n-ary) a number of propagators are considered. In general, each propagator:
  - affects one variable (aiming at narrowing its domain, when invoked);
  - is triggered by some events, namely some change in the domain of some variable;
- For example, the posting of the constraint c :: x + y = z creates 3 propagators

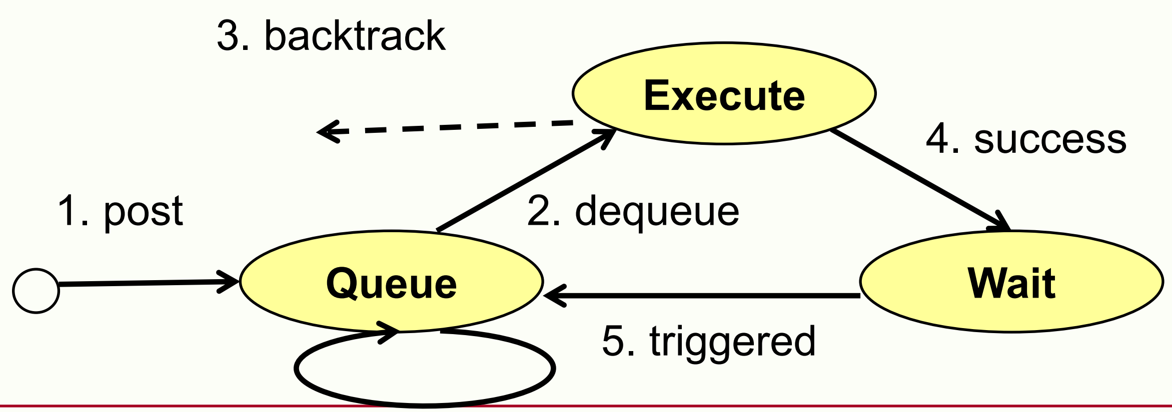
$$\mathbf{P_x: x \leftarrow y - z \quad ; \quad P_y: y \leftarrow z - x \quad ; \quad P_z: z \leftarrow x + y}$$

- Propagator $\mathbf{P_x}$ (likewise for propagators $\mathbf{P_y}$ and $\mathbf{P_z}$) is triggered by some change in the domain of variables y or z.
- When executed it (possibly) narrows the domain of x. If this becomes empty, a failure is detected and backtracks is enforced.

Constraint Propagation

- The life cycle of such propagators can be schematically represented as follows:

1. Propagators are created when the corresponding constraint is posted. They are enqueued and become ready for execution.
2. When they reach the front of the queue they are executed. Upon execution the domain of the propagator variable is possibly narrowed.
3. If the domain is empty, backtracking occurs, and after trailing, the propagator is put back in the queue.
4. Otherwise, the propagator stays waiting for a triggering event.
5. When one such event occurs the propagator is enqueued . While enqueued, other triggering events are possibly “merged” in the queue.



Constraint Propagation

P_x: $x \leftarrow y - z$; P_y: $y \leftarrow z - x$; P_z: $z \leftarrow x + y$

- Propagators aim at maintaining some form of consistency, typically domain consistency or bounds consistency, This has a direct influence on the events that trigger them.
- For example, with bounds consistency, propagator Px is triggered when the maximum or minimum values in the domain of variables y and z is changed. These are the only events that change the maximum and minimum values of the domain of variable x.
- In contrast, if domain consistency is maintained, propagator Px is triggered whenever any value is removed from the domain of any of the variables y or z, since these removals may end the support of some value in the domain of x.
- This also means that sometimes the activation of the propagator does not lead to the removal of any value in the domain. For example value 3 in x may be supported by either values 5 and 2, or by values 7 and 4 for variables y and z. If 7 is removed from the domain of y, x= 3 still has support in y and z.

Generalised arc-consistency: Global Constraints

- The time complexity of generalised arc consistency for n-ary constraints may be too costly. Consider the case of k variables that all have to take different values.

$$x_1 \neq x_2,\ x_1 \neq x_3 \ldots x_1 \neq x_k \ldots x_{k-1} \neq x_k$$

- These k(k-1)/2 binary constraints can be replaced by a single k-ary constraint

all_different(x_1 , x_2, x_3 , .. , x_k)

- However, checking the consistency of such constraint by the naïve method presented, would have complexity **$O(a\ k^2\ d^{k+1})$** , i.e. **$O(k^4\ d^{k+1})$**.
- This is why, some very widely used n-ary constraints are dealt with as **global constraints**, for which special purpose, and much faster, algorithms exist to check the constraint consistency.
- In the **all_different** constraint, an algorithm based in graph theory enforces this checking with complexity **$O(d\ k^{3/2})$**, much better than the naïve version.
- For example for **$d \approx k \approx 9$** (sudoku problem!) the number of checks is reduced from $9^2 * 9^{10} \approx$ **$3*10^{10}$** to a much more acceptable number of $9 * 9^{3/2} \approx$ **243**.