# ECE 566

## Constraint Satisfaction Problems and Techniques

# Outline

(1) Constraint Satisfaction problems.

1.0 Definitions, Constraint network representation, features of CSP's.

(2) CSP problem solving

2.1 Problem reduction

2.2 Consistency, Satisfiability

2.3 Solution Strategies

2.4 Search orders in CSP.

(3) Interval Constraint Satisfaction Problems.

3.1 Interval arithmetic.

3.2 Constraints with Integer, Real and Interval values.

3.3 Interval Constraint networks.

3.4 Waltz Filtering algorithm.

*What is a Constraint Satisfaction Problem(CSP)?*

A CSP consists of :

- Variables - a finite number.
- Domain - finite or infinite domain.
- Constraints - Restricting what values variables can simultaneously take.

  Example: 8 - Queens problem

  Variables: The eight queens' positions.

  Domain: The Chessboard squares.

  Constraints: No queen attacks the other.

*Goal: To find an "Assignment" of variables satisfying all the constraints.*

A CSP is also referred to as a "Consistent labeling problem"

Simple Classification based on the types of variables and constraints:

(1) Discrete CSP's (logical CSP's). e.g. : A planning problem

Variables are symbolic variables, Constraints are logical constraints.

(2)Numerical CSP's. e.g. : a linear programming problem

Variables are numerical, constraints: functions,equations etc.

# Formal definition of a CSP:

CSP is a triple (Z,D,C)

Z: Set of variables.

$Z = \{ x_1, x_2, \ldots . x_n \}$

D: Z -> set of objects of any type.

D maps variables Z to abstract objects in the real world.

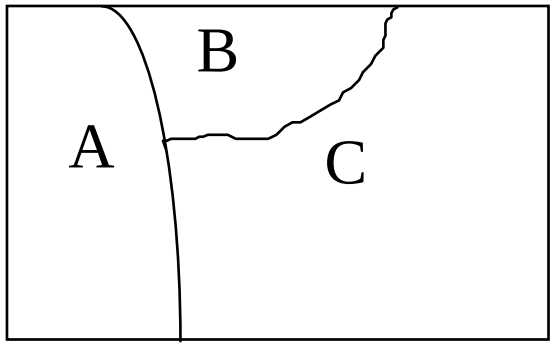
For each variable:

$D_{x_i}$ = The set of possible values for variable $x_i$.

(domain of $x_i$ )

C: Set of Constraints. (Set of sets of compound labels)

# Example of a simple CSP.




Map Coloring Problem.
Colors given:Red,Blue,Green.

Z = A,B,C. (Variables)

$D_i$ = Red,Blue,Green. (For each variable) , i = A,B,C.
Domain of the variables.

C = No two adjacent regions have the same color.
= A != B, A != C, B != C.
(Set of constraints)

Why CSP's?

- Many Design and Engineering problems can be formulated as CSP's.
- Algorithms using special features of a CSP are available for effective solution.

*Examples of problems that can be modeled as CSP's:*

- Machine Vision.(Waltz)
- Job-Shop Scheduling.(Prosser,Fox etc.)
- Graph problems.(Haddock)
- Temporal and Spatial Reasoning.(Tsang)
- Electrical Circuit Design.(Sussman,Steele)
- Mechanical Design.(Serrano,Nudel,Navinchandra etc.)
- Diagnostic Reasoning.(Pearl)
- Constraint programming.(Uses CSP concepts)

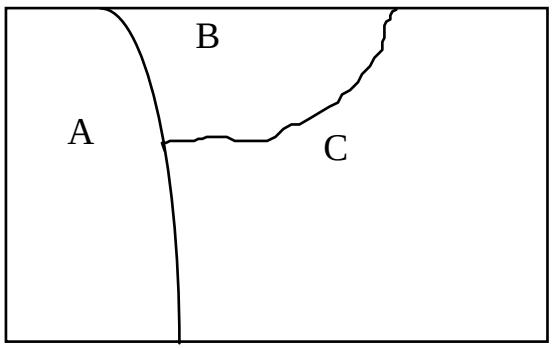
## Constraint Network:

It is a graphical representation of a CSP where Nodes represent variables and Arcs represent constraints.

Binary CSP - Constraint graph.

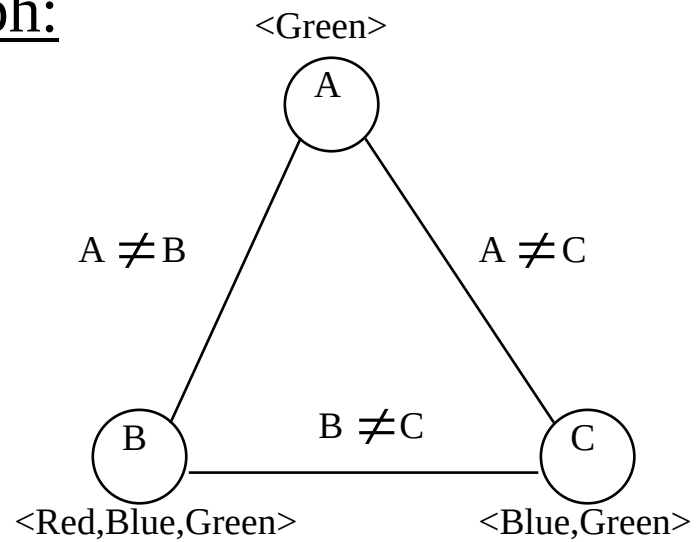
General CSP - Constraint Hyper-graph.

## Constraint Graph:




Map Coloring Problem.
Colors that can be used:
Red , Blue , Green.

Constraints:
Adjacent colors different.
A (Green only)
B (An color)
C (Blue or Green)



Label: variable-value pair assigning a value to the variable.
Notation: L = <x,a>
Example: L1 = < A,Red >
Here variable “A” is assigned the value “Red”.

Compound label: Simultaneous assignment of values to a set of variables.

Notation: CL = ($<x_1,v_1>$ $<x_2,v_2>$. . . . . $<x_n,v_n>$)

Example: CL1=(<A,Red>,<B,Green>,<C,Blue>).

Here a simultaneous assignment is made to three variables A, B and C simultaneously.

k- Compound label: A Compound label assigning values to k variables simultaneously.

Example: CL1 above is a 3- compound label.

Constraints: Sets of legal compound labels for sets of variables.
(Restrictions on what values the variables can simultaneously take.)

Notation: $C_s$ - s may be a variable or a set of variables.
Constraints may be defined with equations, with predicates, with computational procedures (functions), or with sets of legal compound labels

Example :
A constraint in a discrete CSP.
for A not equal B in a map coloring problem assume that domain of A = <Red> and that of B = < Red, Green, Blue>.
we can conceptually visualize this as a set of all legal compound labels
$C_{AB}$= ( (<A,Red>,<B,Green>), (<A,Red>,<B,Blue>) )

Unary Constraints: Constraints on only one variable.
Example: Variable A not equal to “Red”

Binary Constraints: relationship between two variables.
Ex: A is not equal to B

Constraint Expression(CE) of a set of variables S: Set of all the constraints on the set S and constraints on all subsets of S.
Notation: CE(S)
Example: Let a set S = { A,B,C}, Then CE(S) includes all the Unary, binary and ternary constraints.
CE(S) = { $C_A$ , $C_B$, $C_C$, $C_{AB}$, $C_{AC}$, $C_{BC}$, $C_{ABC}$ }

- General CSP: A CSP which is not limited to Binary relationships.
- Lot of research in CSP literature refers to Binary CSP’s.
- (Any CSP can be converted to a Binary CSP)

Satisfiability: Shows us that the problem is solvable and a solution exists.

k-satisfiability:A k-compound label CL satisfies a constraint expression CE if and only if CL satisfies all constraints in CE. In the above example if a 3-compound label is
CL= (< A, Red>, < B, Blue>, <C, Green>) then CL 3-satisfies CE(S) if and only if it satisfies all the unary, binary and ternary constraint in CE(S).

k-satisfiable CSP: For all subsets of k-variables,We can find legal values for all the k-variables.
(Note that the total number of variables in the CSP may be n > k).
Example: Consider a CSP with three variables A, B and C.
It is 2-satisfiable if all constraints are satisfied for all sets of 2 variables, i.e. Constraints on AB, AC, and BC.

Satisfiable CSP: A CSP which has n-variables is “Satisfiable” if it is n-satisfiable (Means that a “solution tuple” exists).

Solution tuple for a CSP: A solution tuple for a CSP is a compound label for all the variables satisfying all the constraints.
Example: for a 3- variable map-coloring problem
solution tuple = (< A, Red>, <B, Blue>, <C, Green>)

*Types of solution for a CSP:*

- One solution.
- All solutions.
- Optimum solutions.

*Types of solution for a CSP:*

- One solution.
- All solutions.
- Optimum solutions.

Features of CSP's:

CSP's can be solved as search problems.

- In discrete CSP's: Size of the search space is finite.

  Number of leaves in the search tree, $L = |D_{X1}| . |D_{X2}| ..... |D_{Xn}|$

  where $|D_{Xi}|$ = Size of the domain of each variable.

  Number of internal nodes: $1 + \blacklozenge |D_{X1}| . |D_{X2}| ..... |D_{Xn}|$

  Ordering affects number of nodes.

*Types of solution for a CSP:*

- One solution.
- All solutions.
- Optimum solutions.

Features of CSP's:

CSP's can be solved as search problems.

- In discrete CSP's: Size of the search space is finite.

  Number of leaves in the search tree, $L = |D_{X1}| . |D_{X2}| ..... |D_{Xn}|$

  where $|D_{Xi}|$ = Size of the domain of each variable.

  Number of internal nodes: $1 + \spadesuit |D_{X1}| . |D_{X2}| ..... |D_{Xn}|$

  Ordering affects number of nodes.

- Depth of the tree is fixed.

*Types of solution for a CSP:*

- One solution.
- All solutions.
- Optimum solutions.

Features of CSP's:

CSP's can be solved as search problems.

- In discrete CSP's: Size of the search space is finite.

  Number of leaves in the search tree, L = $| D_{X1}| . | D_{X2}|..... | D_{Xn}|$

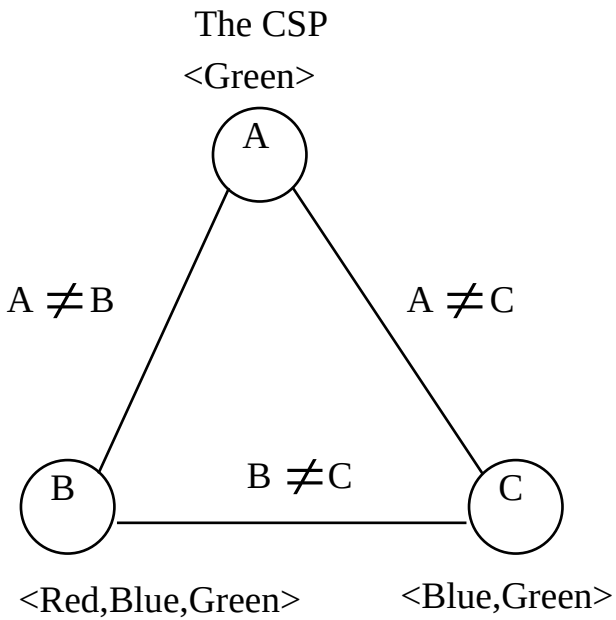
  where $| D_{Xi}|$ = Size of the domain of each variable.

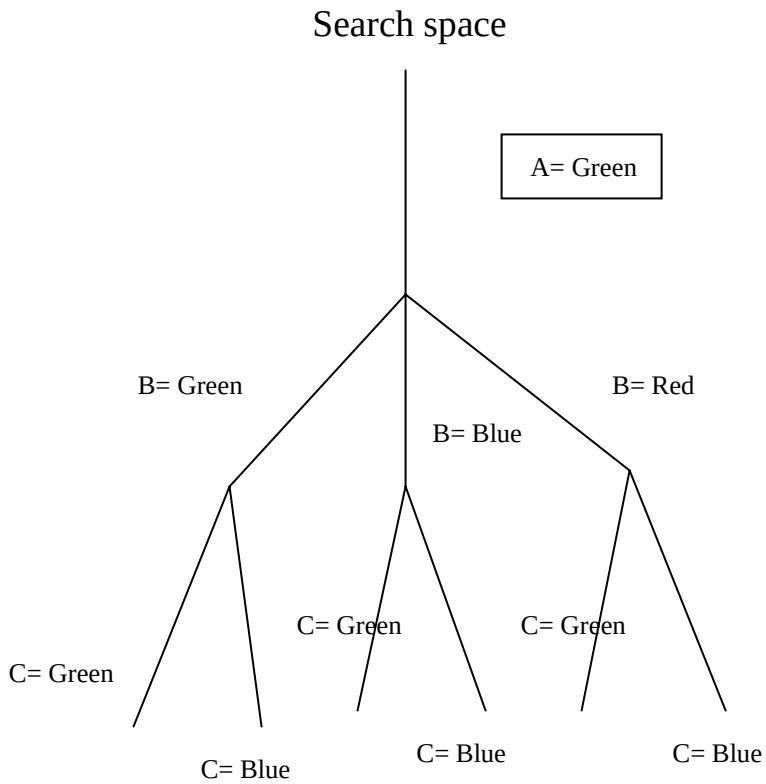
  Number of internal nodes: $1+ ♦ | D_{X1}| . | D_{X2}|..... | D_{Xn}|$

  Ordering affects number of nodes.

- Depth of the tree is fixed.
- Subtrees in the search tree are similar.

# Features of CSP's




The number of leaves = $|D_A| . |D_B| . |D_C| = 1 \times 3 \times 2 = 6$.

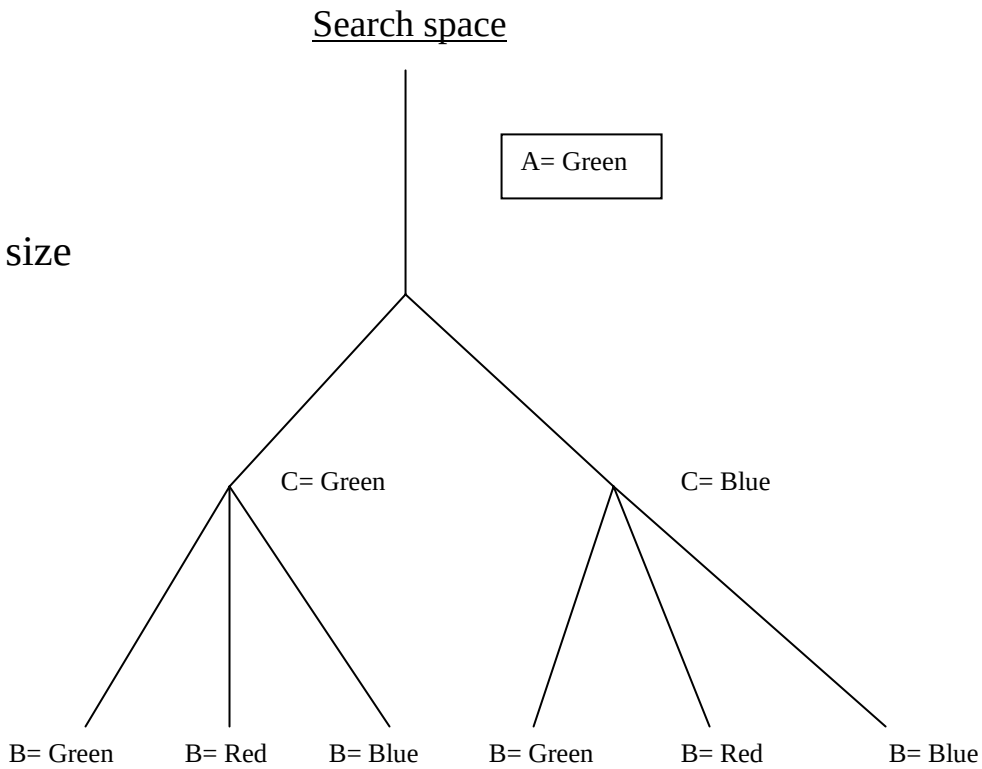
Number of internal nodes: $1 + \blacklozenge |D_{X1}| . |D_{X2}| ..... |D_{Xn}| = 1 + 1 + 1 \times 3 + 1 \times 3 \times 2 = 11$

Ordering: A,B,C.

Features of CSP's, Ordering affects the size of search space.

The order with ascending order of domain size gives the least number of internal nodes.

This is the least for the given problem.




Number of internal nodes: $1+ \blacklozenge | D_{X1}| . | D_{X2}|..... |D_{Xn}|= 1 + 1 +1 \times 2 +1 \times 3 \times 2 =10$

Ordering: A,C,B.

Note: with this ordering the number of internal nodes is 10, In previous case it was 11. But the number of leaves does not change.

## Example of a general search strategy

The simplest algorithm for solving a search problem is the Chronological Backtracking.

## Solution using a general search strategy

The simplest algorithm for solving a search problem is the Chronological Backtracking.

The algorithm works as follows:

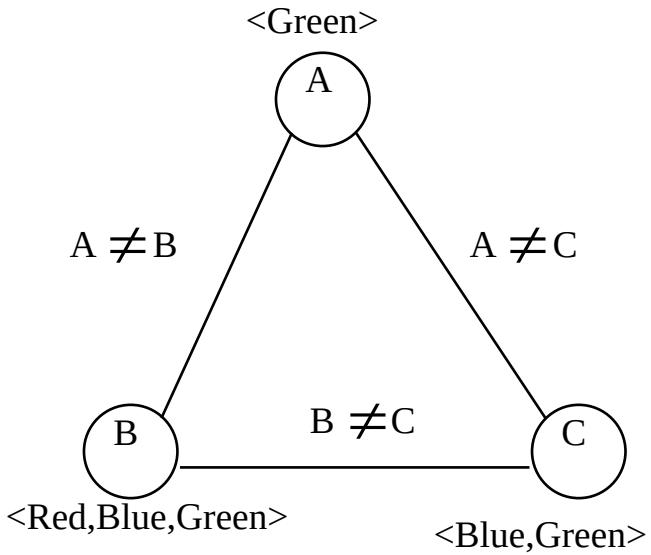
(1) Pick a variable from the set of variables.

(2) Assign a value for the variable from the domain to form a compound label. If the compound label satisfies all constraints, go to (3), else backtrack and pick another value

If no value can be picked, then problem has no solution.

(3) Check if all the variables are labeled. If yes,Then the compound label is the solution tuple. If no, then go to (1).

# Example of Chronological Back tracking:




Unlabeled pick value compound label

(1) {A, B, C} A Green {<A,Green>}

No constraints violated , Pick another variable.

(2) {B,C} B Green {<A,Green>, <B,Green>}

Constraint violated, *Backtrack* and Pick another value for B

(3) {B,C} B Red {<A,Green>, <B,Red>}

No constraints violated , Pick another variable.

(4) {C} C Green {<A,Green>, <B,Red>, <C,Green>}

Constraint violated, *Backtrack* and Pick another value for C.

(5) {C} C Blue {<A,Green>, <B,Red>, <C,Blue>}

No constraints violated and no more variables unlabeled.

So the compound label is the solution.

Backtrack free search: If a solution can be found using the backtracking algorithm without any backtracking.

Definition: A search is backtrack free in an ordering if for every variable that is to be labeled , we can find a value for it compatible with all the variables assigned before.

In the previous example we saw there were instances where backtracking occurred.(Steps 2 and 4).

The temporal complexity of the algorithm is $O(ea^n)$ and the spatial complexity is O(log n) where n is the number of variables, e is the number of constraints and a is the domain size.

Problems with the simple backtracking algorithm:

Inefficient: More operations than necessary.

Reasons:

**(1) Local view of the problem.(Trashing behavior)**

In the algorithm, attention is focussed on the present variable and not the problem as a whole.Consider n variables $x_1$ ….$x_n$. Suppose we have assigned values for variables $x_1$ to $x_{n-1}$. Suppose that we assign a certain value to $x_n$ that is incompatible with the first variable $x_1$ ,the algorithm will backtrack to the previous variable $x_{n-1}$ and try out all values, and so on till it goes back and assigns a different value for $x_1$.Many backtracks avoidable - If knowledge that $x_1$ is causing problem is available.

**(2) No learning property:** Consider variables $x_i$, $x_j$ and $x_n$. Assume that when we assign values,we first assign values to $x_i$ ,then to $x_j$ and then to $x_n$.If some value of the domain in $x_n$ is incompatible with $x_j$, then this will be detected when we first assign values to $x_j$, and try to assign values to $x_n$. Assume that due to some backtrack we have to go back to $x_i$, Now when we assign some value to $x_j$, we also have to assign values to $x_n$ , the algorithm does not remember that a certain value of $x_n$ was incompatible with a value of $x_j$, the algorithm checks for the particular value in $x_n$ that was causing the problem, ***again.***

Both these concepts can be easily understood by the example which follows.

Example:

Consider a simple CSP consisting of the three variables x,y,z whose domains are respectively

Dx={4,5,6} Dy ={1,2} and Dz={2,3}

The Constraints given are x+y+z > 10 x+z > 8 and y+z >4

Consider the trace of a backtracking algorithm:

| Step | x | y | z | Cxyz | Cyz | Cxz |
|---|---|---|---|---|---|---|
| 1 | 4 | 1 | 2 | No | No | No |
| 2 | | 1 | 3 | No | No | No |
| 3 | | 2 | 2 | No | No | No |
| 4 | | 2 | 3 | No | Yes | No |
| 5 | 5 | 1 | 2 | No | No | No |
| 6 | | 1 | 3 | No | No | No |
| 7 | | 2 | 2 | No | No | No |
| 8 | | 2 | 3 | No | Yes | No |
| 9 | 6 | 1 | 2 | No | No | No |
| 10 | | 1 | 3 | No | No | No |
| 11 | | 2 | 2 | No | No | No |
| 12 | | 2 | 3 | Yes | Yes | Yes |

We assume that the back-tracking algorithm chooses the variables in this order i.e x, y and z.

C xyz denotes constraints on x,y and z and similarly Cxz and Cyz denote constraints on X,Z and Y,Z respectively.

“No” signifies that constraint is not satisfied.

(1) For local view observe that we can only satisfy the constraint x+z > 8 only if x > 5 , But the backtracking algorithm doesn’t use this fact, backtracks 10 times before it finds the solution.

(2) For the no learning property consider the constraint y + z > 4. Note steps 1 to 4, we discovered that only the value of y=2 and z =3 will satisfy this property. But when we backtracked to x and changed x to 5, we never use this fact and carry out the backtracks for y and z (Steps 5 to 7) and discover this again.

## Overview of Problem reduction:

- Transform original CSP to new equivalent problem.
  (Equivalent problem: A problem with same sets of variables and solution tuples as the original problem).
- Problem may be easier to solve.
- Easier to recognize insoluble problems.

A preprocessing step, It is rare that a solution can be obtained by only problem reduction.

## Reduction process:

- Modify Domains: Remove redundant values.
  Redundant - Don't appear in "*any*" solution tuple.
- Tighten Constraints: Fewer compound labels satisfy constraints , i.e. remove redundant compound labels.

Ex:

(1) $D_X$={1,2,3,4,5}. If constraints are x < 4 and x < 5.
Value 4,5 are redundant. Also we note that the Constraint x< 5 is redundant.

(2) Example for redundant compound label: Consider three variables, A,B and C. Assume that binary constraints exist between each of these variables.
Consider a compound label (<A,a>,<C,c>) satisfying constraints on A and C,
This can be a redundant compound label, If we cannot assign any value b to B such that (<A,a>,<B,b>) or (<B,b>,<C,c>) can be satisfied.

Gains from reduction:

- Reduction in search space.
- Avoiding repeatedly searching futile sub-trees.
- Detect insoluble problems.

How to detect Redundant values?
By using the Consistency techniques.

Minimal Problem:

- No Redundant Values in the domain and No redundant Compound labels in the constraints.
- Reduction to a Minimal problem is a NP-hard problem.

Fundamental Concepts in CSP:

How to detect redundant values and compound labels for problem reduction ?

- Consistency Techniques

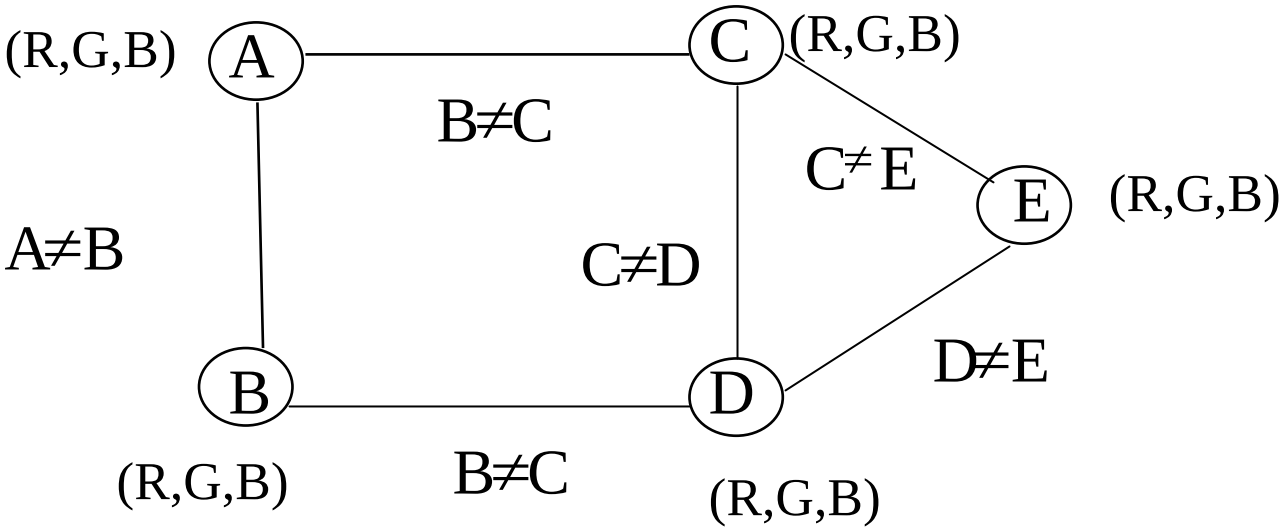
  -defined such that presence of redundant values and redundant compound labels falsifies consistency requirements. Neither a Sufficient nor a Necessary condition for a problem to be solvable.

Consistency:

1 - Consistency: Every value in every domain satisfies Unary Constraints on the variable.

k - Consistency: For all (k-1) compound labels satisfying the constraints we can add an additional label to form a k-compound label satisfying all the relevant constraints.

## Example for k-consistency:




(R,G,B) are the values that the variables can take, red, green and Blue.

We can show that this problem is 5-consistent, i.e for all 4-compound labels satisfying all the constraints, we can add another label to get a 5-compound label satisfying all the constraints.

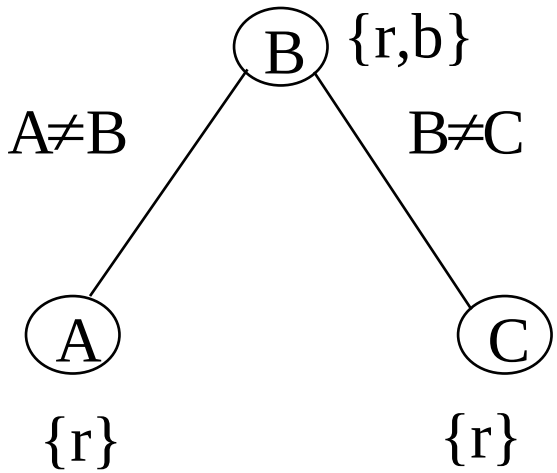
# K-consistency continued.

The table below shows the 4-compound labels(values) for the variables A,B, C and D, It shows that we can select a value for E in the domain to make a 5-compound label satisfying all the constraints.Thus it is 5-consistent.

The table shows the situation when A takes the value Blue, we can show that the same situation applies when A takes Red or Green, and also when we consider the other variables and assign alternate values to them.

| A | B | C | D | | E |
|---|---|---|---|---|---|
| (Variables) | | | | | |
| Blue | Red | Red | Green | | Blue |
| Blue | Red | Green | Blue | | Red |
| Blue | Red | Red | Blue | | Green |
| Blue | Green | Red | Blue | | Green |
| Blue | Green | Green | Red | | Blue |
| Blue | Green | Green | Blue | | Red |

# k - Consistency does not imply k-1 Consistency.




3-consistent CSP but not 2-consistent.

In the above example: problem is 3- consistent because for the 2-compound labels which are consistent (<A,r>,<C,r>), (<A,r>,<B,b>) (<B,b>,<C,r>)

We can add another label(<B,b>, <C,r> and <A,r> respectively for the above 2-consistent labels.) to get a 3-compound label satisfying the constraints.However it is not 2-consistent( we can never use <B,r> to get a 2-compound label satisfying the constraints).(Recall definition of 2-consistency.)

Strong k-Consistency : CSP is 1,2,3..... k-Consistent

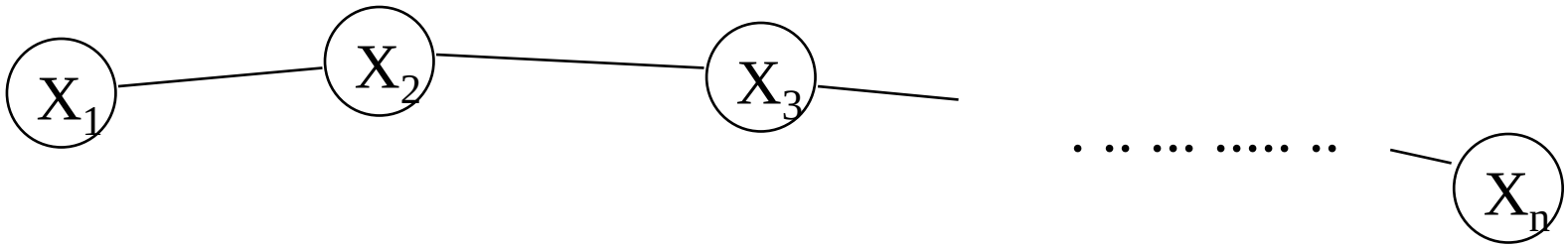
## Special Consistency definitions for Binary CSP's:

- Node Consistency:Each possible node value satisfies the constraint involving only that variable.(Same as 1-Consistency).

- Arc Consistency: Consider an arc(x,y),The arc is arc-consistent if for every value of x in its domain,We can find a value for y satisfying the constraint

- A CSP is arc-consistent if all its arcs are arc-consistent. (Same as 2- Consistency)
  Node consistency and Arc consistency can be used to remove redundant values in the domain.

## Path Consistency:




For a compound label (<$x_1$,$v_1$>,<$x_n$,$v_n$>) satisfying all constraints on $x_1$ and $x_n$, we can find values for all the variables ($x_2$, $x_3$…… ,$x_{n-1}$) in the path satisfying the binary constraint for each arc in the path.A CSP is path-consistent if all its paths are path-consistent.

Directional Arc and Path Consistency:
Here the variables are ordered.

(Removal of Redundant Values and Constraints)

Node Consistency: removes redundant values.

Arc Consistency: stronger condition than node consistency removes more redundant values.

Path Consistency: stronger condition than Arc consistency not only removes redundant values, but also redundant compound labels.

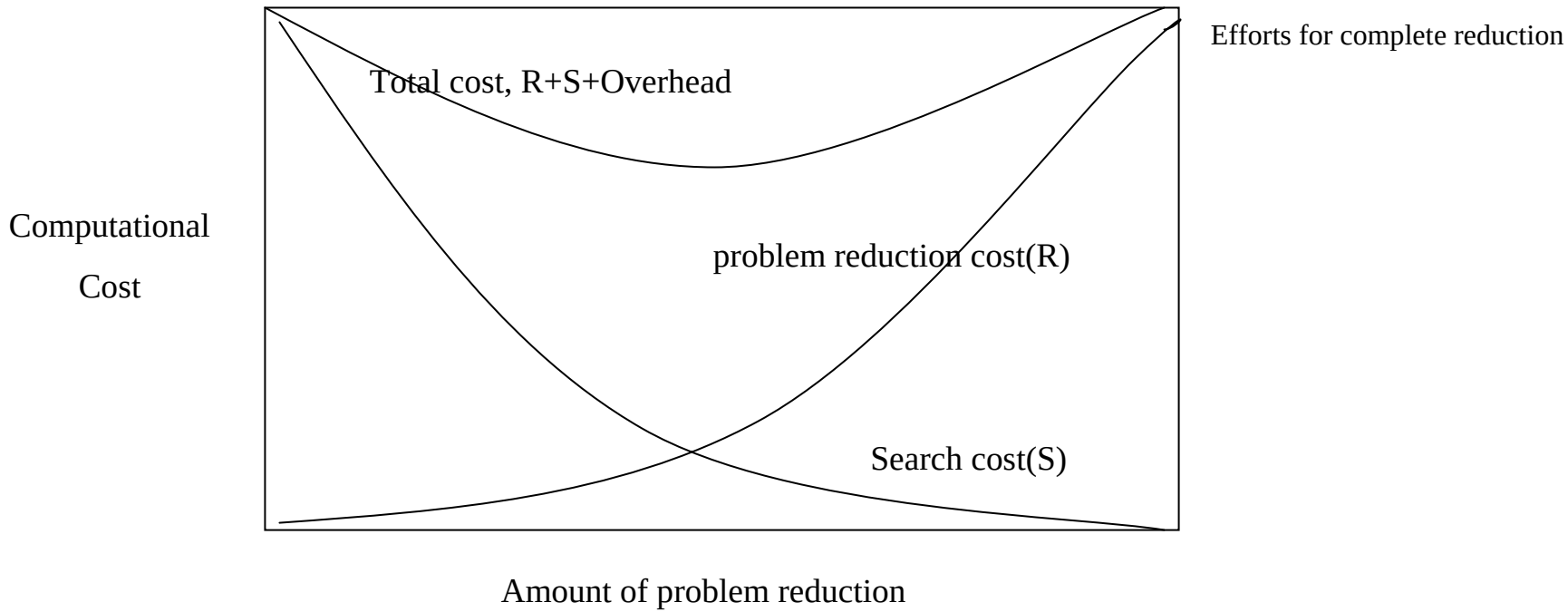
Why do reduction at all?

The time complexity of reduction is less in many cases than searching and this offers good benefits in problem solving over using just searching without problem reduction.

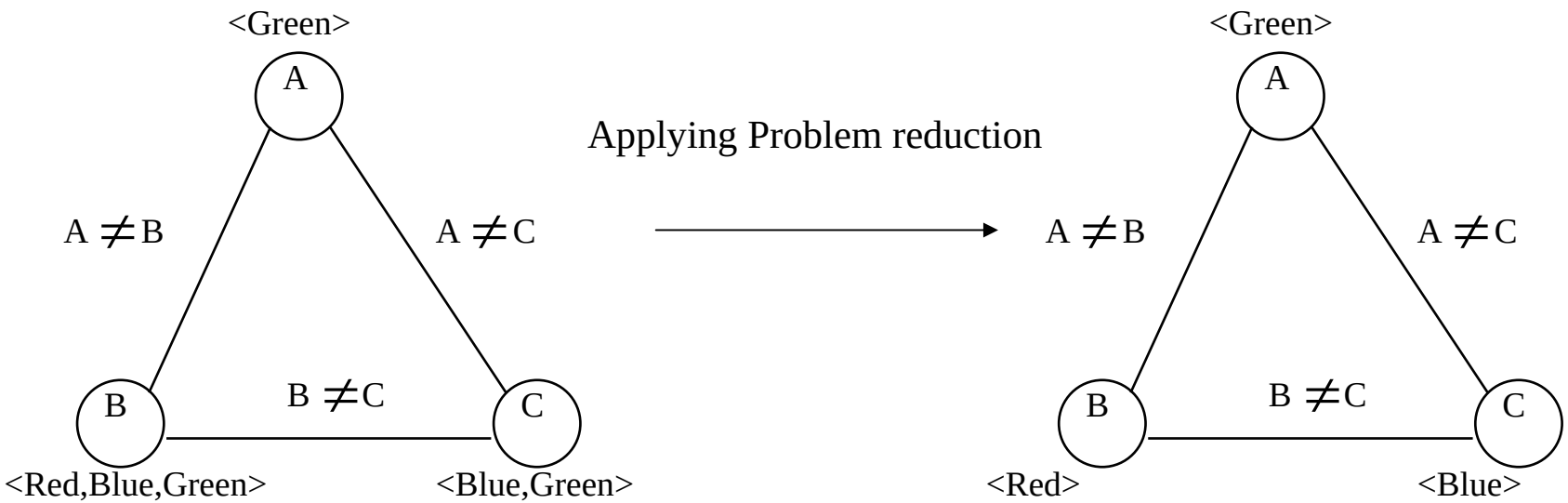
# Problem Reduction Cost Savings

Remove redundant values from domains,tighten constraints without losing solution tuples.

Problem Reduction ⟶ Equivalent problem



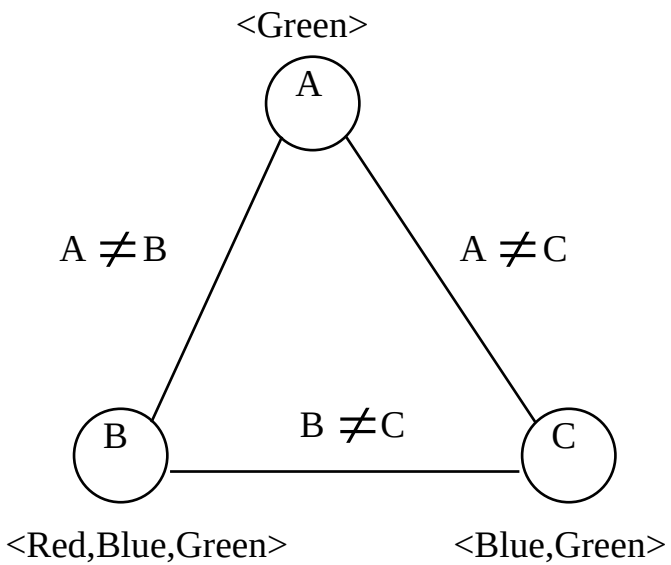
# Example to show Problem reduction:



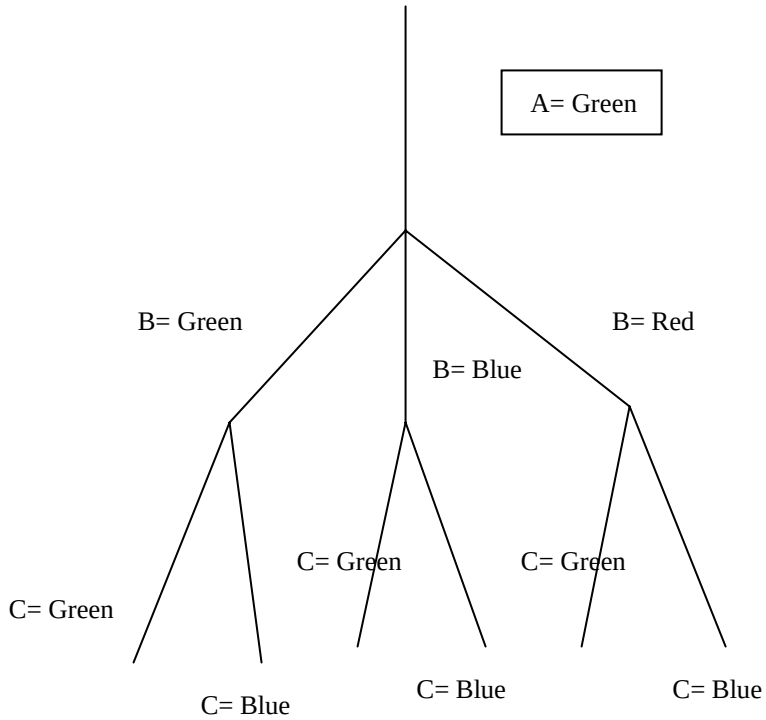

Redundant values in variable domain removed

# How search efficiency is increased by problem reduction?

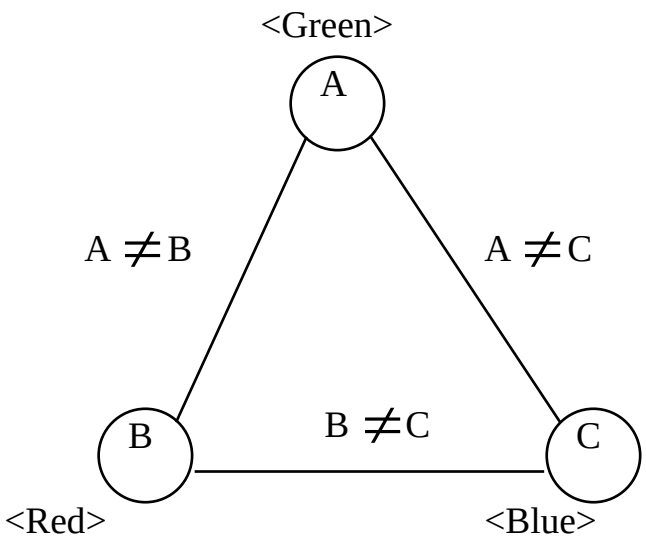
The Original problem




Original Search space

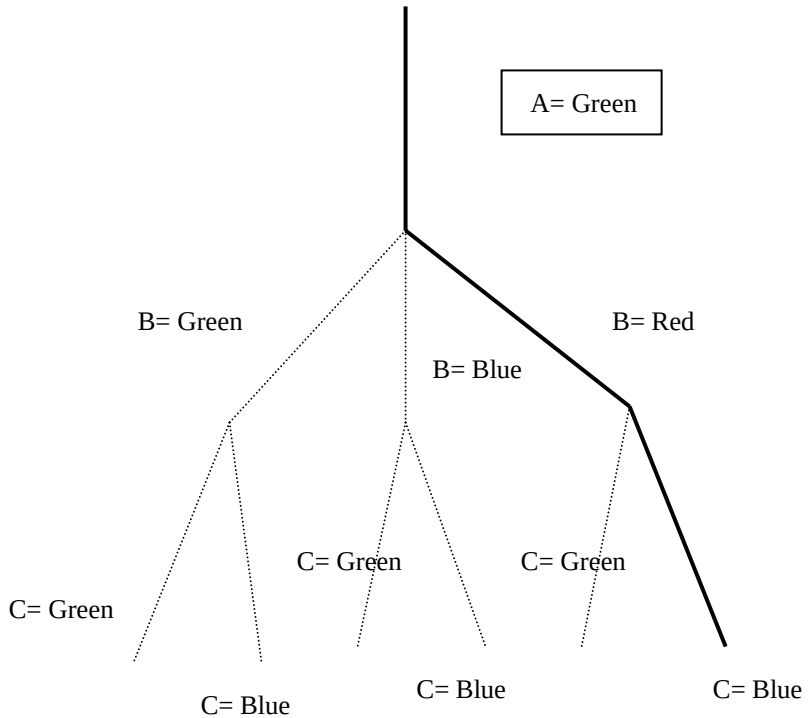


# The problem after reduction




Redundant values in variable domain removed

Reduced Search space




The bold lines indicate the new- search space,the dotted lines are pruned.

The reduced problem here is backtrack free

## An Algorithm for Arc-Consistency achievement

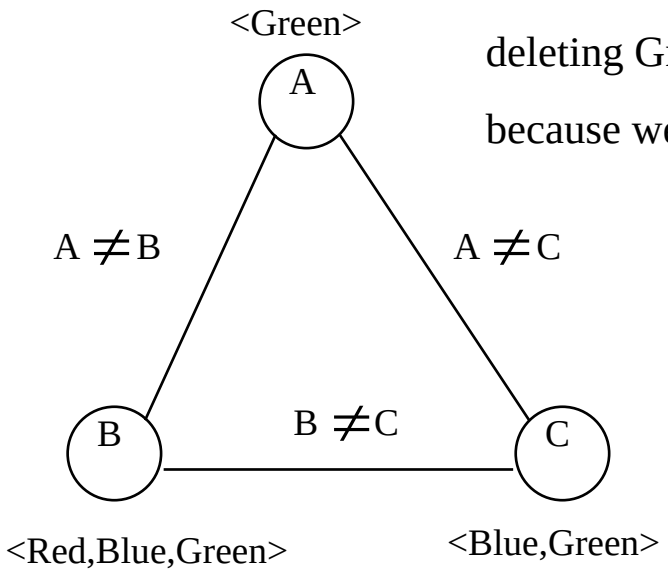
An arc ( $V_i$, $V_j$) can be made consistent by simply deleting values from the domain of $D_i$ ($V_i$) for which we cannot find consistent values in the domain of the variable $V_j$.The following algorithm which does that will be used as a part of the algorithm for Arc-consistency.

Algorithm REVISE

```
procedure REVISE(  (Vi , Vj), (Z,D,C) )
        DELETE < -  false;
        for each X in Di do
            if there is no such Vj in Dj such that ( X, Vj) is consistent,
                then
                delete X from Di ;
                DELETE <- true;
            endif ;
        endfor ;
return DELETE;
end REVISE
```

We have to note that when we revise the domain of a variable $V_i$,then each previously revised arc ($V_k$, $V_i$) has to be revised since some of the values in the domain of $V_k$ may no longer be compatible, since we may have deleted some values from $V_i$ .

Example:




Consider this problem which was discussed earlier, B and C are initially consistent, But when we make A and C consistent by deleting Green from the domain of C, B and C become inconsistent because we do not have a value for C when B is assigned value Blue.

Here's an algorithm called AC-3 which uses algorithm REVISE( ) to achieve arc-consistency in an entire graph.

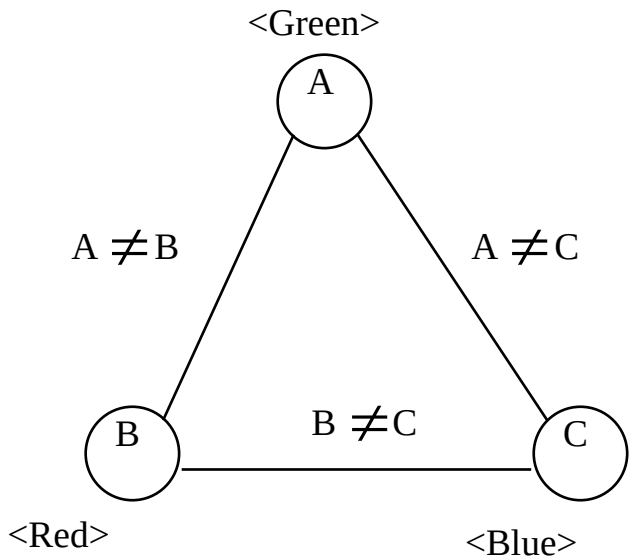
```
procedure AC-3 (Z,D,C)
    Q <- { (Vi ,Vj ) ∈ arcs(G), i ≠ j };
      while Q not empty
            select and delete any arc ( Vk ,Vm ) from Q.
            if (REVISE( (Vk ,Vm ),(Z,D,C)) then
            Q ∪{ (Vi ,Vk ) such that (Vi ,Vk ) ∈ arcs(G), i ≠k, i ≠ m }
            endif;
      endwhile;
end AC-3
```

We note that we use the Revise( $(V_i, V_{j,})$ , $(Z,D,C)$ ) procedure defined earlier in this algorithm.

## Working of the AC-3 algorithm

A list of arcs Q is maintained, We remove one arc ( $V_k$, $V_m$) from the list and choose a value X in the domain of $V_k$ , and see if there exist values for $V_m$ such that the constraint on this arc is satisfied, If no such value is available, we delete X from the $D_k$. The above steps are carried out by the REVISE algorithm. When any values in the domain of $V_k$ are changed, we include into the list Q, arcs that may be possibly affected by this deletion, i.e. All the arcs that are connected to $V_k$ , ($V_i$,$V_k$ ) are added.

Various revisions of the above algorithm are available, These are called AC-1, AC-2 and AC-4. Similarly for Path-Consistency we have PC-1, PC-2, PC-3 and PC-4. [Foundations of Constraint Satisfaction, E.Tsang, 1993]




It is easy to verify that using the above algorithm modifies the original problem to the arc-consistent problem shown here. Here we can now solve the problem without any back-tracking, But this is not so in general. (Arc-consistency is not a strong enough condition to always eliminate back-tracking)

## Comparison between the various algorithms:

Algorithms: (T-Time complexity and S-Space complexity)

(1) Node Consistency algorithm- just removes values not-satisfying unary constraints. T=O(an), S= O(an).

(2) Arc Consistency algorithms(achieves node consistency also)

AC-1,AC-2, AC-3, AC-4 (improvements in temporal efficiency).

AC-1, $T=O(a^3ne)$, S= O(e+an).

AC-3, $T=O(a^3e)$, S= O(e+an).

AC-4, $T=O(a^2e)$, S= $O(a^2e)$.

(3) Path Consistency algorithms:

PC-1, $T=O(a^5n^5)$, S= $O(n^3a^2)$.

PC-2, $T=O(a^5n^3)$, S= $O(n^3+n^2a^2)$.

PC-4, $T=O(a^3n^3)$, S= $O(n^3a^3)$.

a = domain size.

e = number of constraints.

n = number of variables.

Relationship between Consistency and Satisfiabilty:

- k-Consistency is insufficient to guarantee satisfiability in a CSP with more than k-variables.
- Path Consistency is not a necessary condition for satisfiability.

Satisfiability Theorem:
A CSP which is 1-satisfiable and strong k-consistent is k-satisfiable for all k.

Theorem due to Freuder: (Condition for back-track free search).
A Binary CSP is backtrack free if the constraint graph forms a tree and both node and arc consistency are achieved.