Artificial Intelligence:
Search Methods for Problem Solving

Constraint Processing
Constraint Satisfaction Problems

A First Course in Artificial Intelligence: Chapter 9

Deepak Khemani

Department of Computer Science & Engineering

IIT Madras

Search vs. Reasoning

So far ...

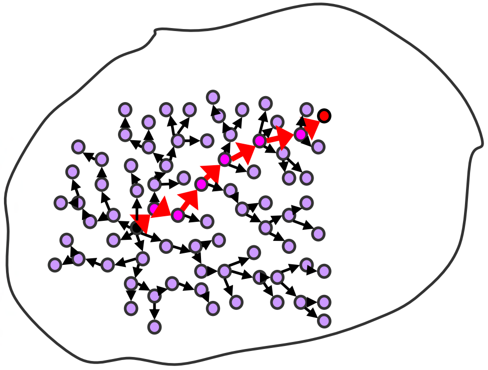
Search

– state space, solution space

– planning problems, configuration problems

– satisfaction, optimal solutions

– trial and error!

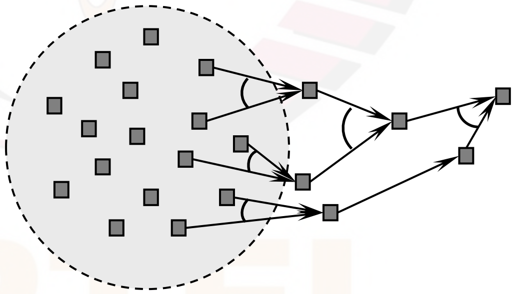


Reasoning

– representation in logic

– entailment and proof

– drawing inferences!



A Unifying Formalism

Constraint Satisfaction Problems (CSPs) are a unifying formalism
that allow a large class of problems
to be represented in a uniform manner,
that allows both
search methods as well as reasoning
to be used for problem solving.

Moreover, search and reasoning can be interleaved.

The user has only to express the problem as a CSP
And an off-the-shelf solver can be used to solve it

Relations *a quick revision*

A mathematical relation on a set of variables is a subset of the cross product of the variables

Domain D = {1, 4, 7, 9} LessThan $\subseteq$ D x D

LessThan = {<1,4>, <1,7>, <1,9>, <4,7>, <4,9>, <7,9>} Extension form

LessThan = {<x,y> | x $\in$ D, y $\in$ D, x < y} Intension form

Domain Courses = {AI, DBMS, ML} Rooms = {24, 26, 36} CourseRoom $\subseteq$ Courses x Rooms

CourseRoom = {<AI, 26>, <DBMS, 24>}

Domain D = {amy, arun, anil, ayesha}

$Aunt_2$ = {<amy, ayesha>, <arun, ayesha>, <anil, ayesha>}

$Brother_2$ = {<amy, arun>, <arun, anil>, <anil, arun>}

$ThreeSiblings_3$ = {<amy, arun, anil>, ... permutations}

Relations form the basis of predicate logic

Constraint Satisfaction Problems

A preview of the course
AI: Constraint Satisfaction Problems

A CSP is a triple <X, D, C>

Also called a Constraint Network or simply a Network $\mathcal{R}$

$$\mathcal{R} = \langle X, D, C \rangle$$

where,

X is a set of variable names

D is a set of domains, one for each variable

- we will confine ourselves to discrete finite domains

C is a set of relations on a subset of variables

- the subset is called the Scope of the relation

Finite Domain Networks

Finite domains can be represented in *extensional* form. For example,

$X = \{Course, Slot, Room, Faculty\}$

$D = \{D_{course}, D_{slot}, D_{room}, D_{faculty}\}$

$D_{course} = \{AI, DBMS, ML, DM\}$

$D_{slot} = \{A, B, C, D, E, F, G\}$

$D_{room} = \{CS24, CS26, CS34, CS36\}$

$D_{faculty} = \{DK, PSK, MK, CSK, JS\}$

$C = \{R_{CS}, R_{CR}, R_{CF}\}$ Binary Constraint Network

$R_{CF} = \{\langle AI, DK\rangle, \langle DM, JS\rangle, \langle ML, PSK\rangle, \langle ML, JS\rangle, \langle PL, PSK\rangle\}$

$R_{CR} = \{\langle AI, CS24\rangle, \langle DM, CS34\rangle, \langle ML, 26\rangle, \langle PL, CS24\rangle\}$

$R_{CS} = \{\langle AI, C\rangle, \langle AI,D\rangle, \langle DM, D\rangle, \langle ML, A\rangle, \langle PL, B\rangle\}$

$R_{FS} = \{\langle DK, C\rangle, \langle DK, F\rangle, \langle MK,D\rangle, \langle DM, D\rangle, \langle ML, A\rangle, \langle PL, B\rangle\}$

Can have other constraints, like no consecutive slots for a faculty...

Solutions

A solution of a CSP

is an assignment of values for *all* the variables

such that *all* the constraints are satisfied

For the previous example this could be,

Course = AI, Slot = C, Room = CS26, Faculty = DK

A network $\mathcal{R}$ is said to express a *solution relation* $\boldsymbol{\rho}$

$\boldsymbol{\rho} = R_{CSDF}$ is a relation on all variables

$\boldsymbol{\rho}$ = set of all possible solutions

(each solution is allocation of one course)

Course Allocation

Finite domains can be represented in *extensional* form. For example,

$X = \{AI, DBMS, ML, DM\}$ note: each course is a variable

$D = \{D_{AI}, D_{DBMS}, D_{ML}, D_{DM}\}$

$D_{AI} = \{DK, MK, SC\}$ the domain is the set of faculty who offer the course

$D_{DBMS} = \{MK, PSK, JS\}$

$D_{ML} = \{DK, PSK, CSK, JS, MN, NSN, CC, AM, NVK, HAM, SD, RR, SC, DJ, CR, KS, RN\}$

$D_{DM} = \{PSK, MK, MN, JS\}$

$C = \{R_{AIML}, R_{DBMSML}\}$ what about slots, rooms, clashes...?

$R_{AIML} = \{\langle DK, JS\rangle, \langle DK, NSN\rangle, \langle MK, CC\rangle, \langle SC, HAM\rangle\}$

$R_{DBMSML} = \{\langle MK, AM\rangle, \langle JS, RR\rangle, \langle PSK, SC\rangle\}$

Alternatively, we can have a universal constraint – $R_{AllDifferent}$

which says each variable (course) has a unique value (faculty)

EXERCISE: Express $R_{AllDifferent}$ in extensional form

Courses, Slots, and Timetables

Consider the course allocation problem

Given - a set of courses
- relations between courses and teachers – who teaches what?
- relations between courses and batches – who can register?
- relations between slots, courses and batches – no clashes
- relations between faculty, slots and slots – no consecutive slots
- relations between courses, slots and rooms – no clashes
- relations between courses, faculty and slots – no clashes

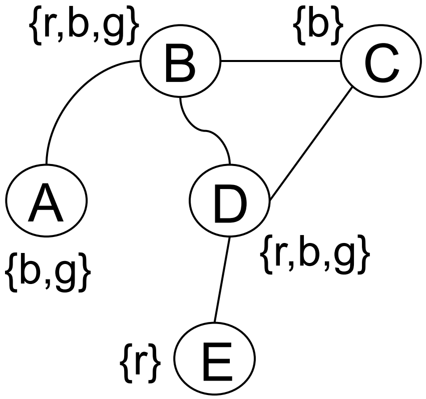
Task: Do course allocation, and slot and room timetabling

Possible additional constraints – DK will teach AI
– DBMS must be in C slot, etc...

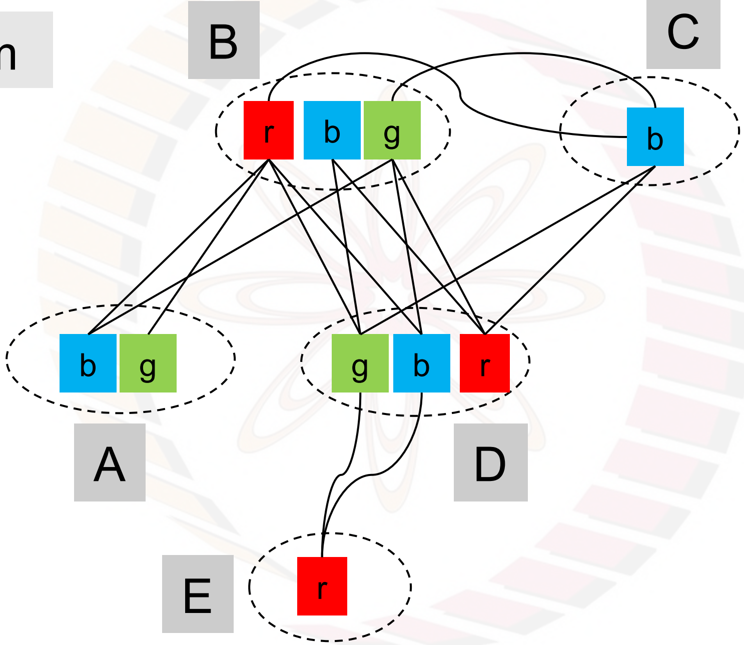
Complex problem – has a separate conference!

The Constraint Graph & the Matching Diagram

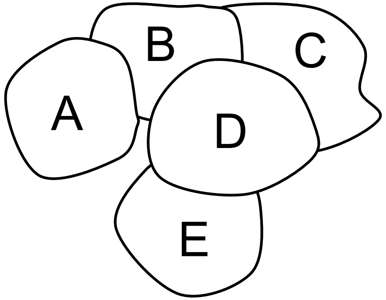
A map colouring problem



The constraint graph



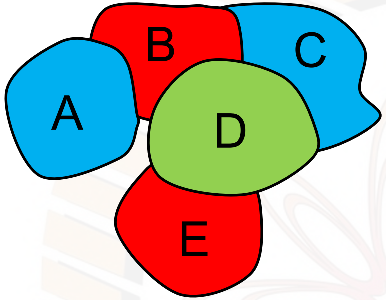
The matching diagram

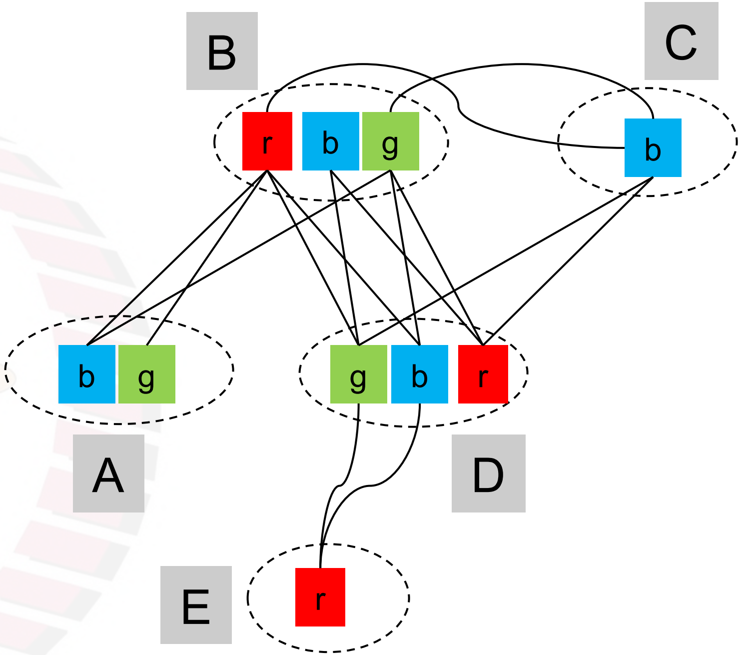


For regions that are not connected the matching diagram has an implicit universal relation. Any combination of values is allowed.

Solutions

Find a solution





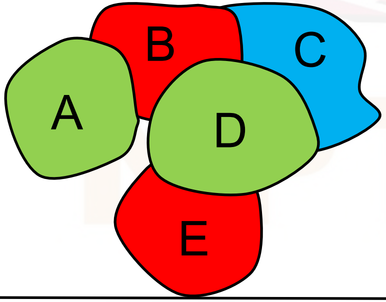
Is there a colouring in which B = b?

No

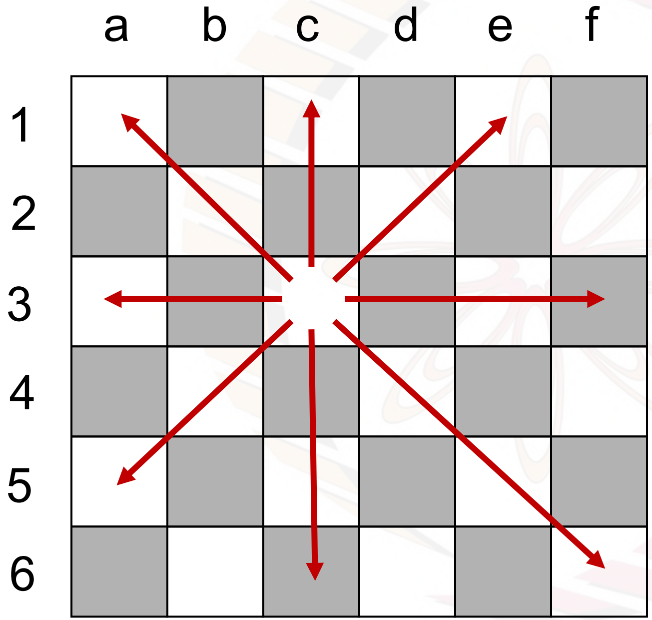
Is there a colouring in which B = g?

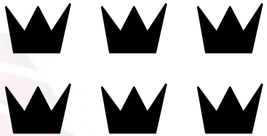
No

Is there a colouring in which A = g?



The 6-Queens problem





The N-Queens problem is to place the N queens on a NxN chess board such that no queen attacks another.

CSPs and Solutions

A CSP describes solutions in parts

- a fog of possibilities
- like the four blindfolded men feeling an elephant
- A BCN for n-Queens describes possible ways of placing two queens

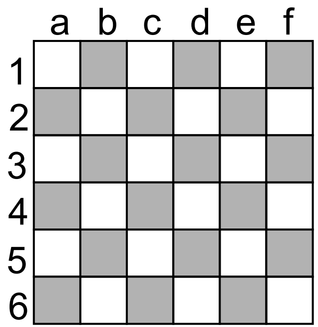
The CSP *expresses* one or more solutions

- the solutions are some valid assignments to variables
- a relation on all the variables of the CSP
- in 6-Queens the solution relation is R_{abcdef}

Solving the CSP is extracting a solution

- an assignment for every variable
- such that all constraints are satisfied
- clearing the fog

n-Queens: Binary Constraint Network



Let us look at 6-Queens

Variables: one variable for each of the 36 squares

Domains: {Q, nil}

Constraints: one binary constraint $\{R_{XY}\}$

R_{XY} = {<a1=Q,a2=0>, <a2=Q,a1=0>, ..., <f6=Q,a1=0>}

constraint – pair of locations where two queens *cannot* be placed

Variables: {a, b, c, d, e, f, g} columns

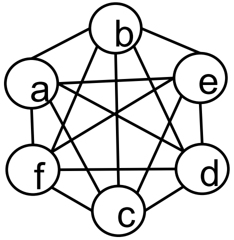
Domains: {1, 2, 3, 4, 5, 6} rows

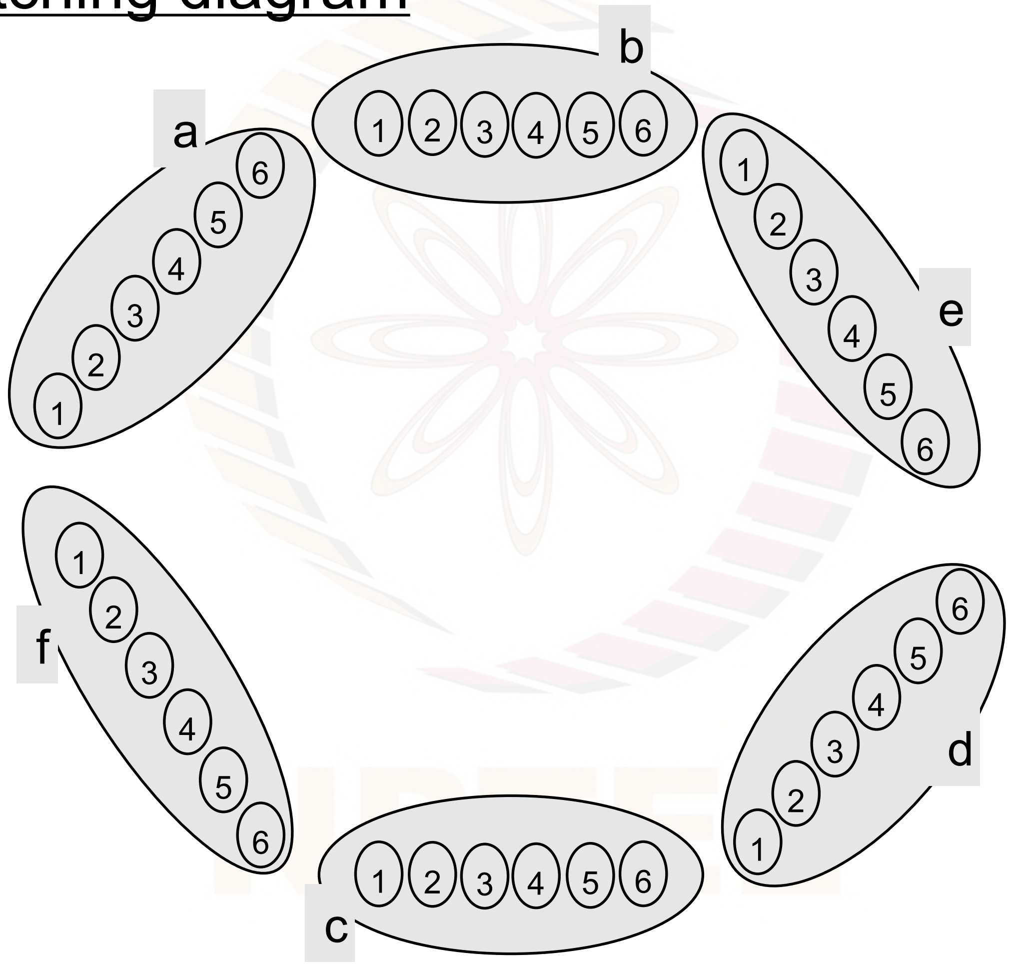
Constraints: $\{R_{ab}, R_{ac}, ..., R_{fg}\}$ pairs of columns

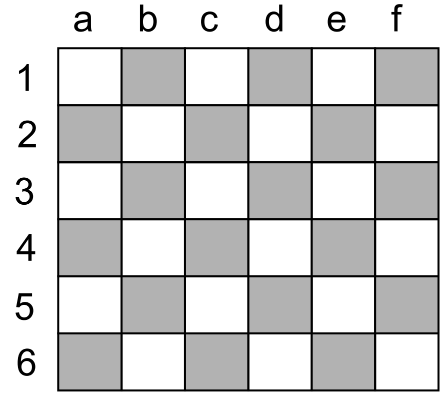
R_{XY}: *Allowed* rows of queens in columns X and Y

R_{ab} = {<1,3>, {<1,4>, <1,5>, <1,6>, <2,4>, ... ,<4,6>}

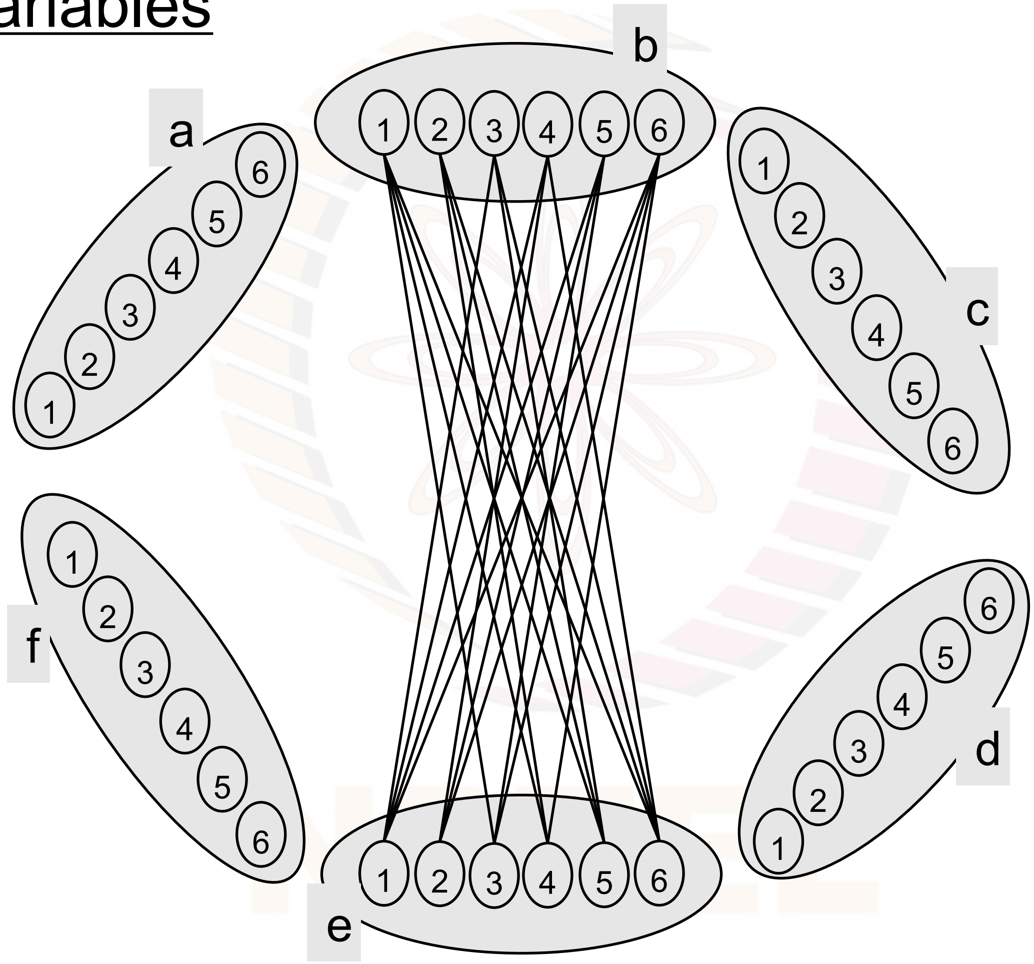
6-Queens: Matching diagram

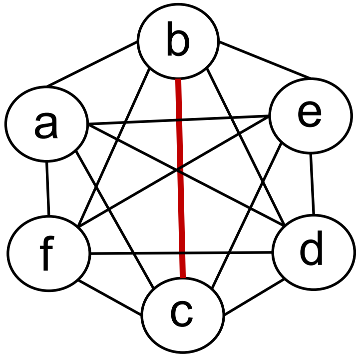




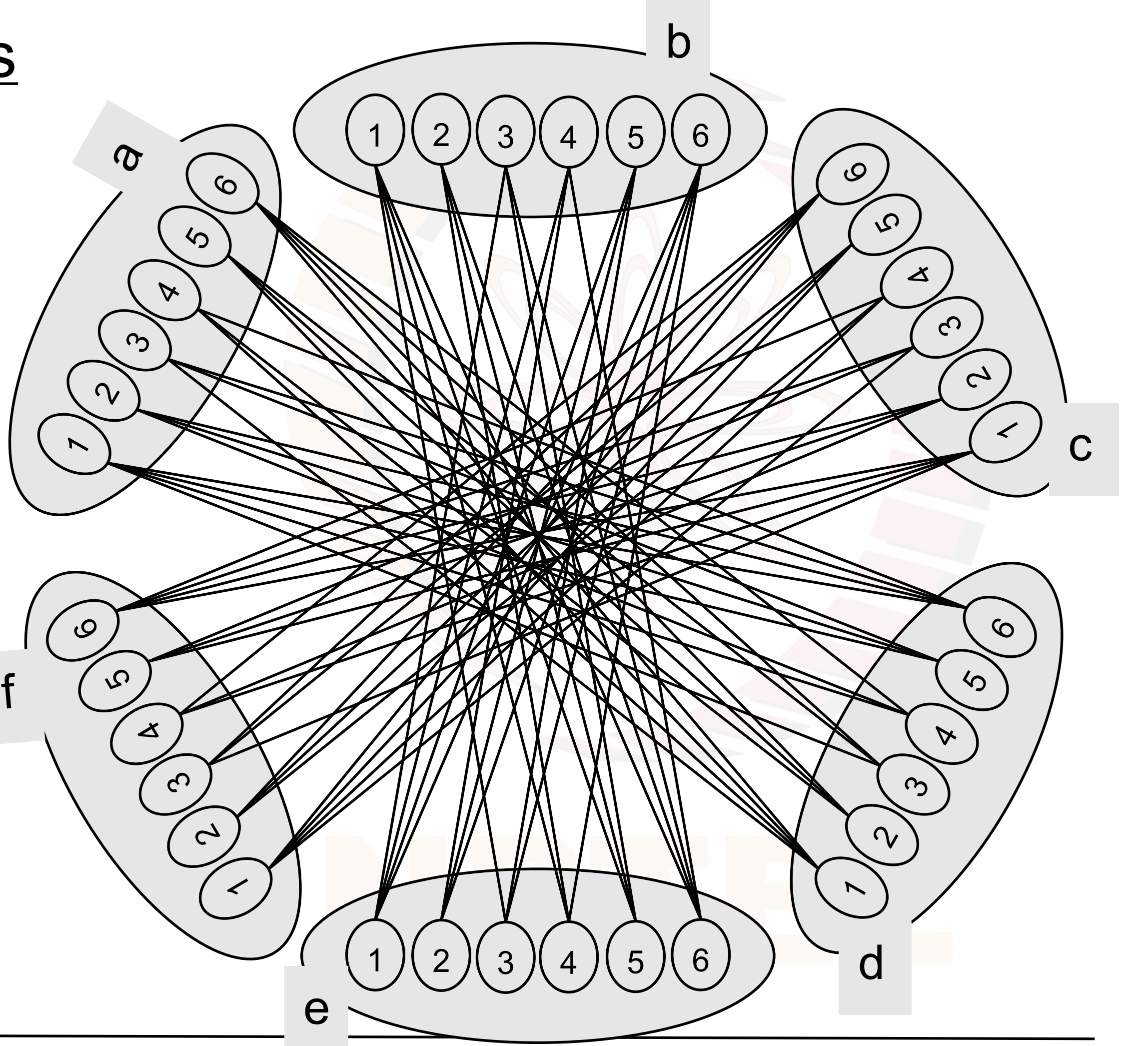


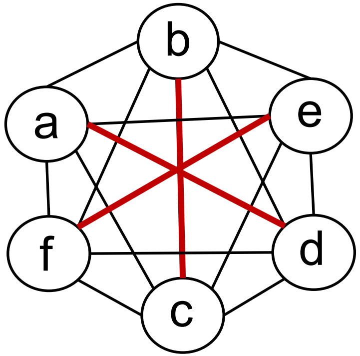
One pair of variables





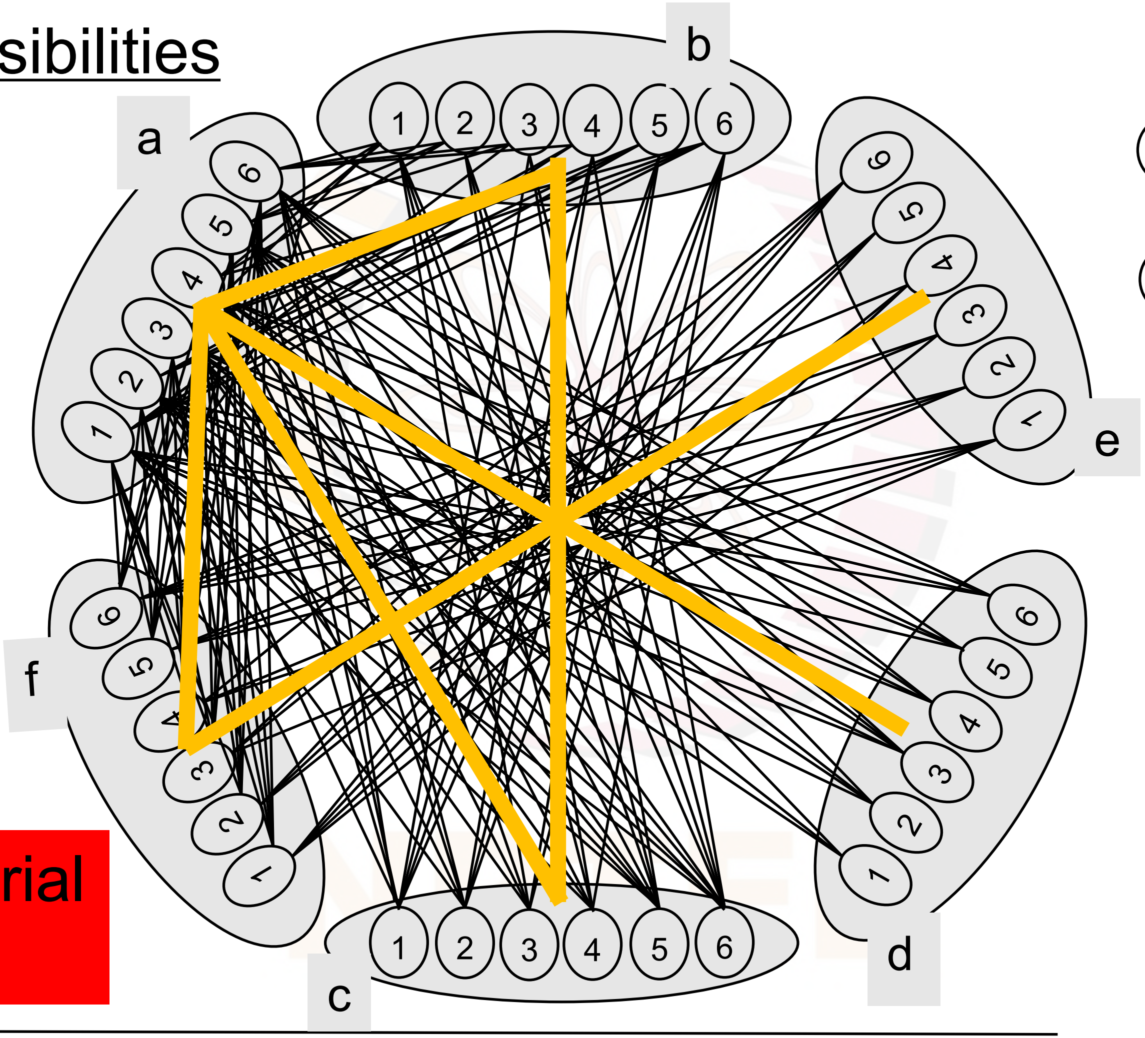
Three pairs

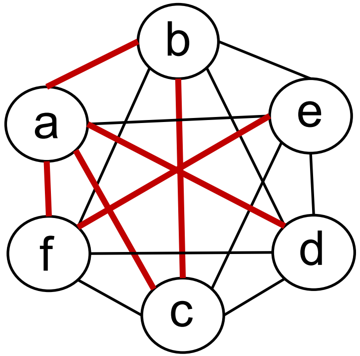




A fog of possibilities

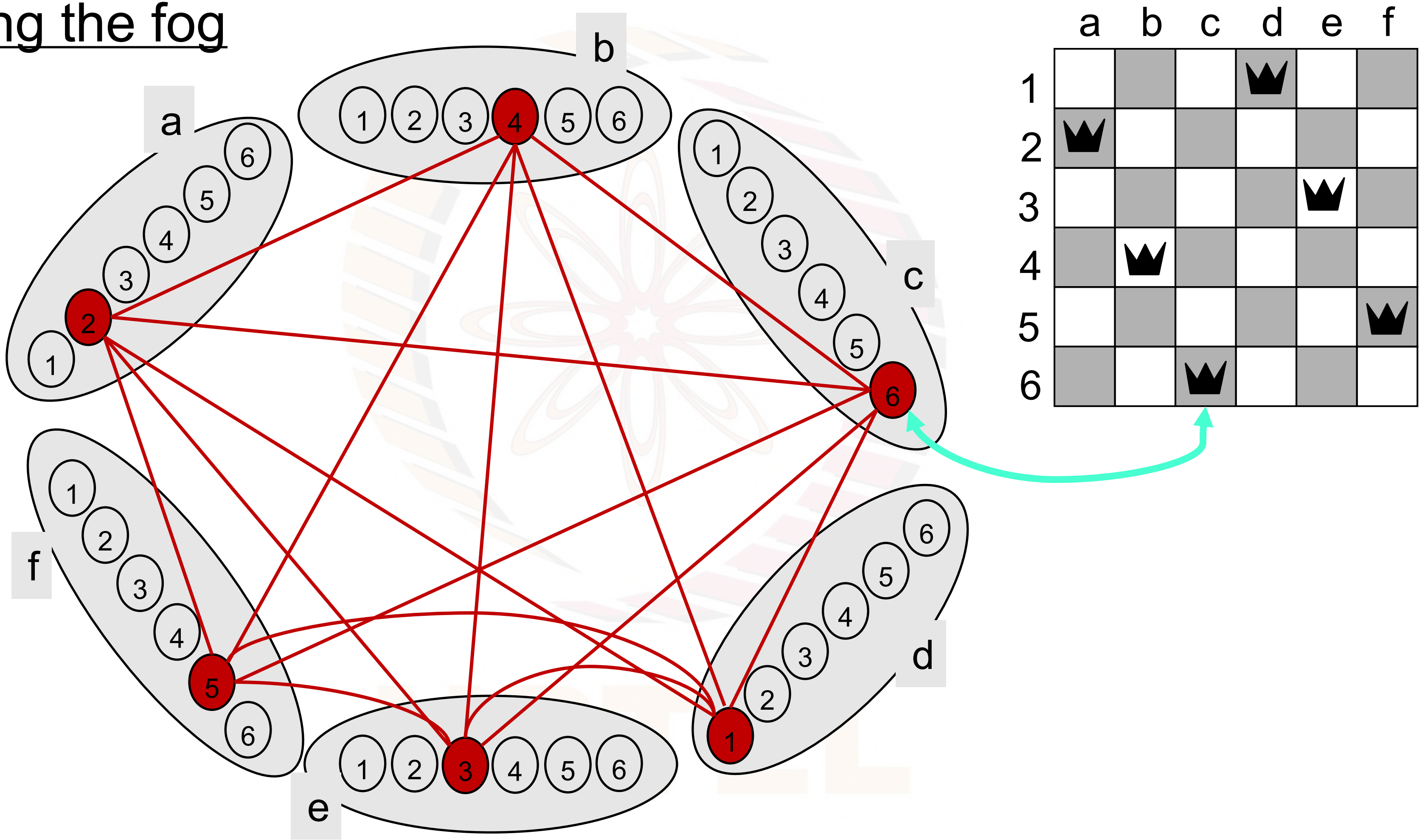
Each edge suggests matching values for two variables





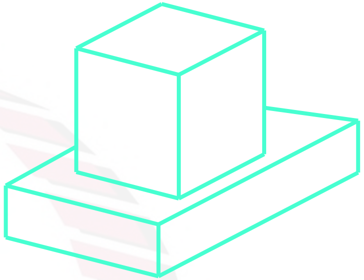
Combinatorial Explosion

Clearing the fog

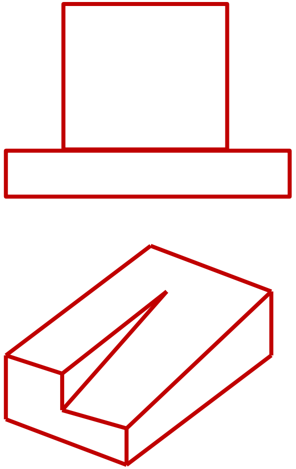


Interpreting line drawings

- Early work in computer vision
- Guzman → Huffman → Waltz
- Huffman
  - trihedral objects (only)
  - 3 planes/edges meeting at a vertex
    - vertices = junctions
    - edges = lines
  - only 18 types of junctions
  - no shadows
  - no cracks
  - normal position (no exceptional viewpoints)
- Huffman's goal was to check if
  a line drawing represented a valid trihedral object



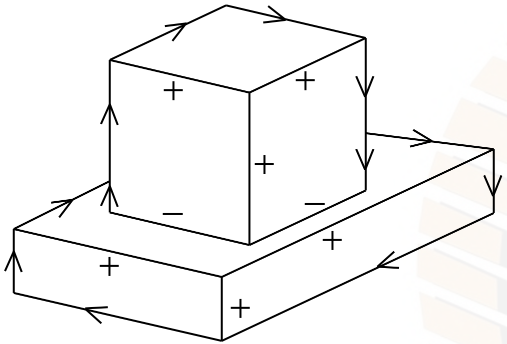
✓



✗

A video by Patrick Winston "Constraints: Interpreting Line Drawings" on YouTube

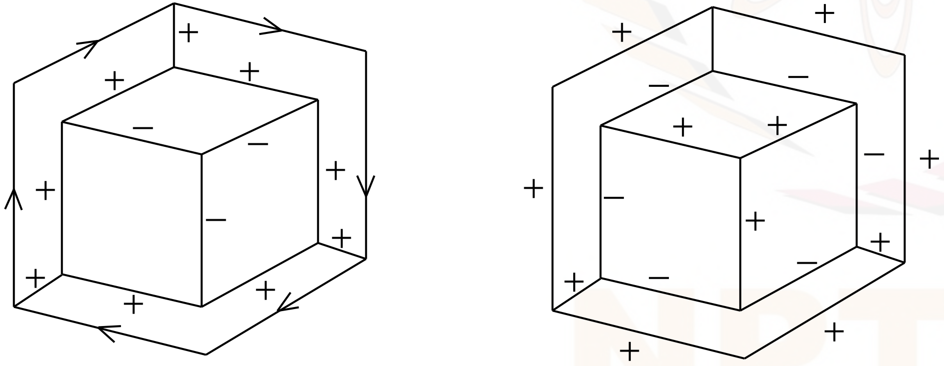
4 types of edges



Arrow - material on right

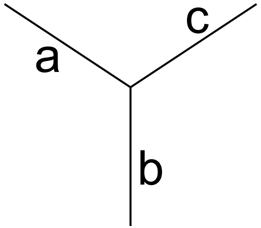
\+ - Convex edge

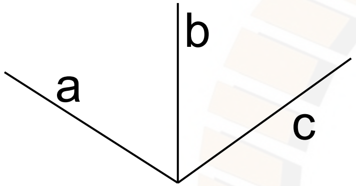
– - Concave edge

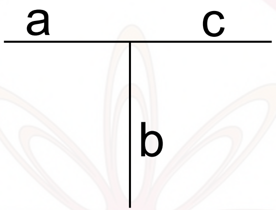


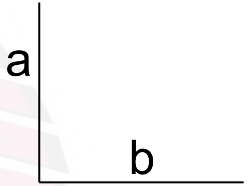
Ambiguous

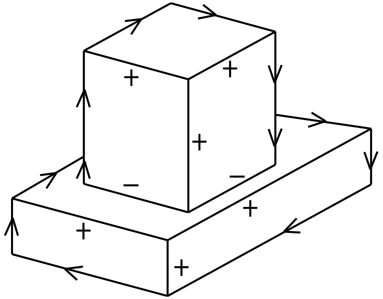
Only 18 types of Vertices











Y joint or Fork

(a,b,c)

(+,+,+)
(–,–,–)
(–,←,←)
(←,–,←)
(←,←,–)

W joint or Arrow

(a,b,c)

(+,–,+)
(–,+,–)
(←,+,←)

T joint

(a,b,c)

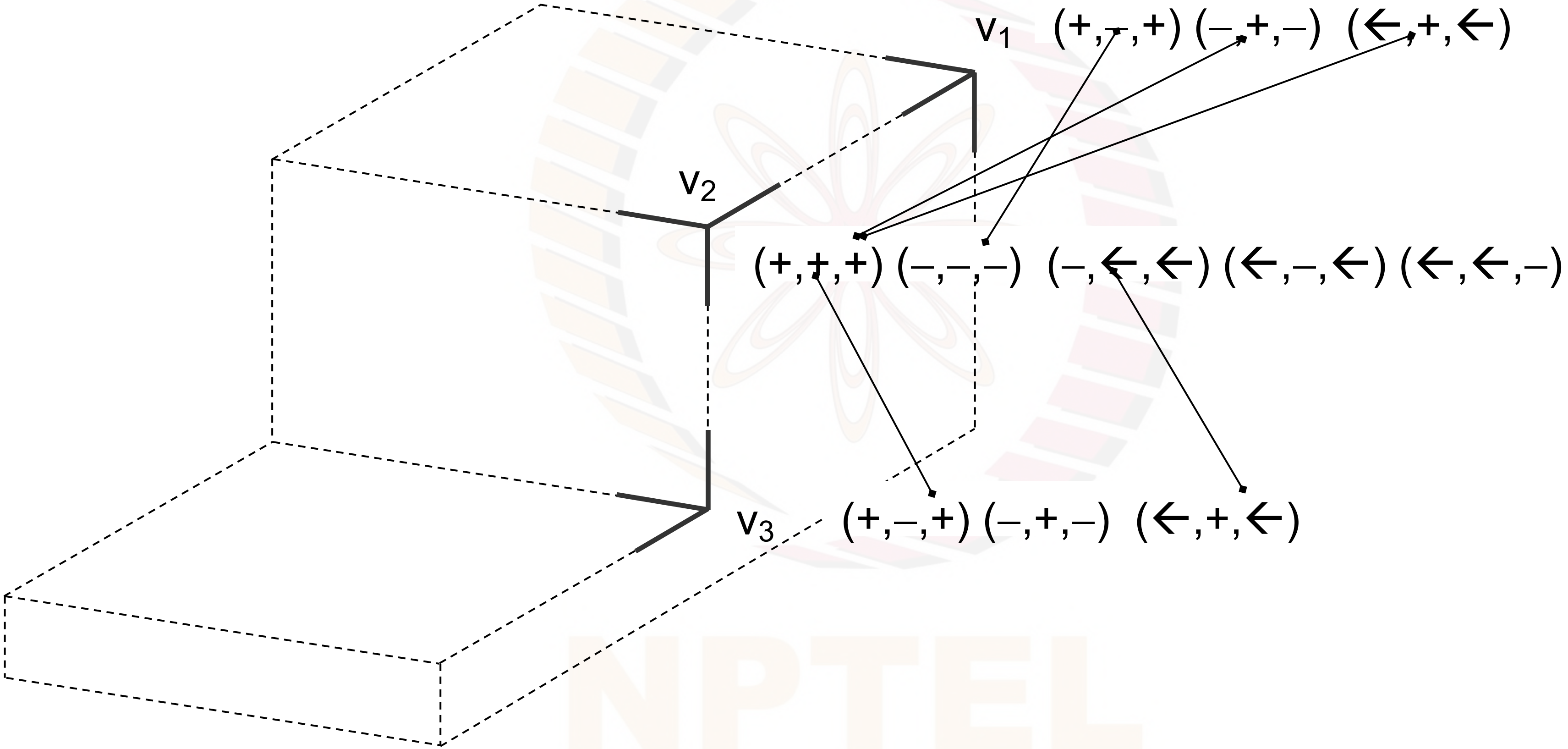
(←,←,←)
(←,→,←)
(←,+,←)
(←,–,←)
(→,+,→)
(→,–,→)

L joint

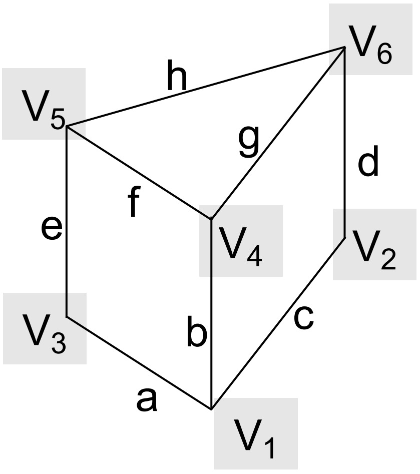
(a,b)

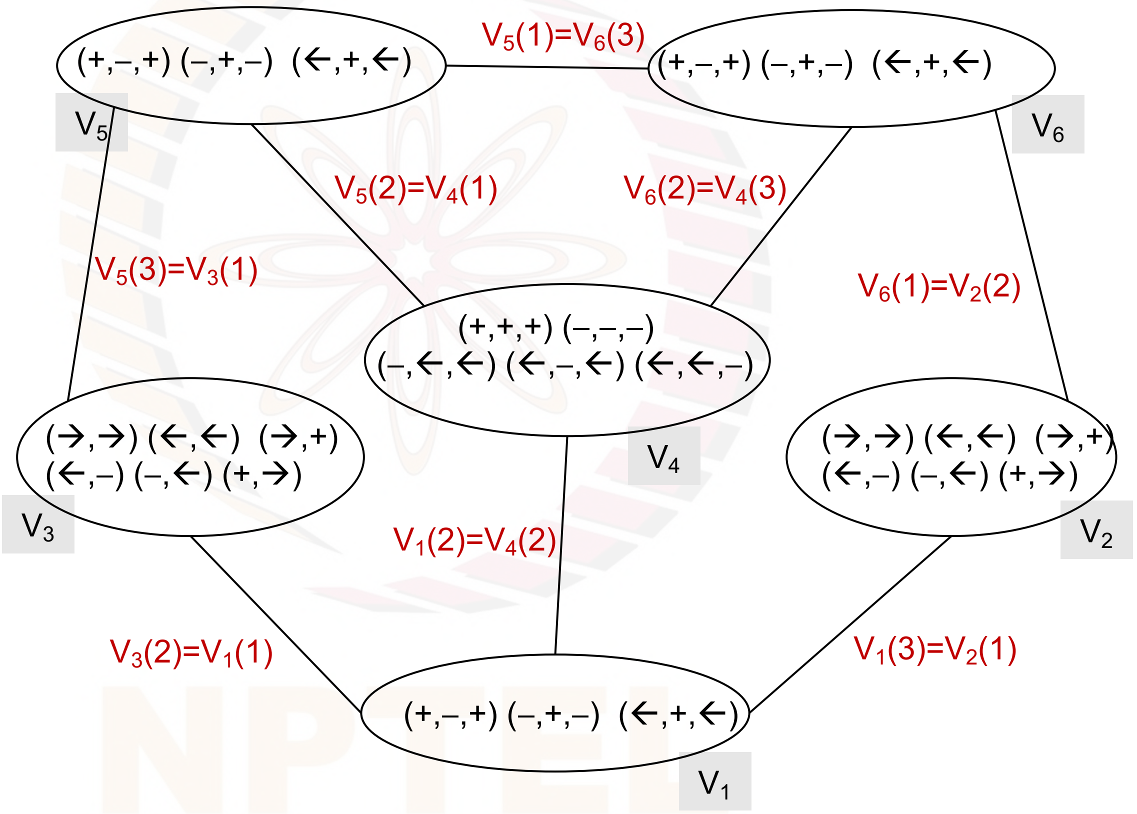
(→,→)
(←,←)
(→,+)
(←,–)
(–,←)
(+,→)

Edge constraints: same label at both ends



A network for a simple block





1 2 3

W vertex: V_1= (a,b,c)
L vertex: V_2= (c,d)
L vertex: V_3= (e,a)
Y vertex: V_4= (b,f,g)
W vertex: V_5= (h,f,e)
W vertex: V_6= (d,g,h)

Watch this space

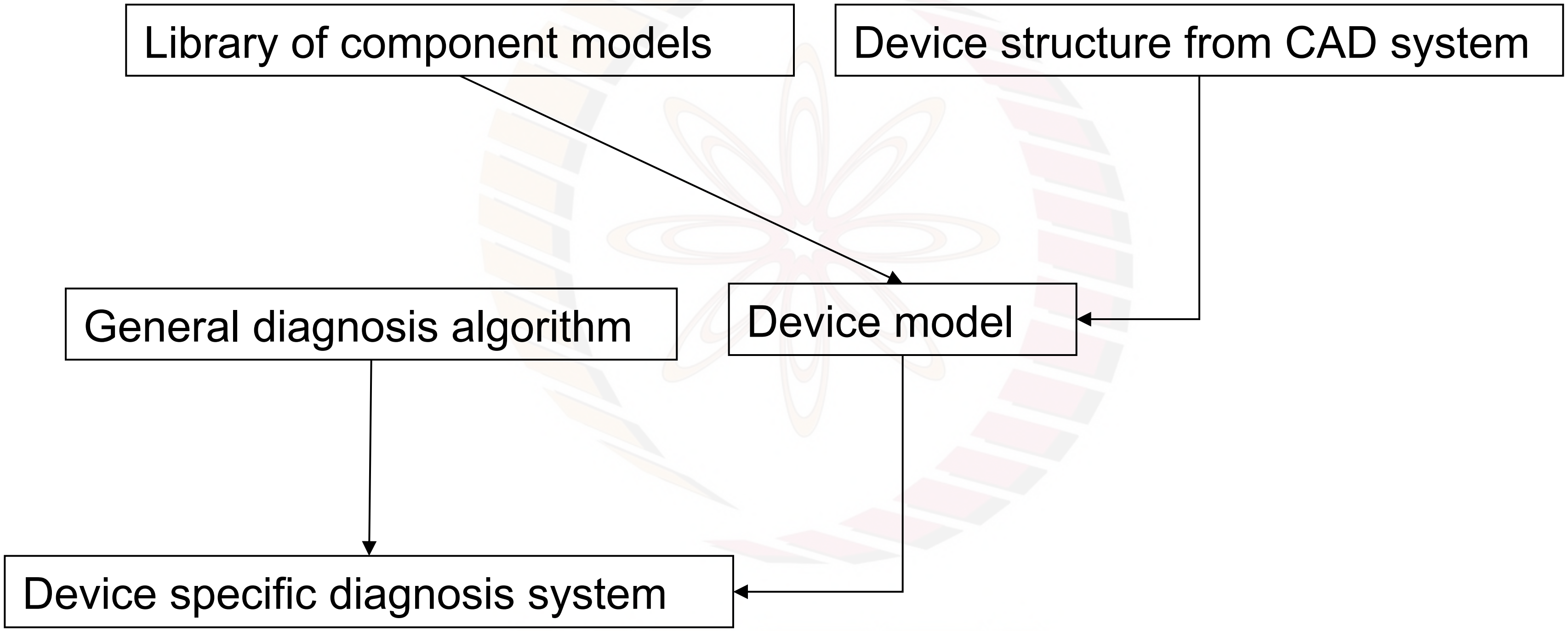
Coming soon....

An efficient algorithm to label line drawings

But first,

Another interesting application...

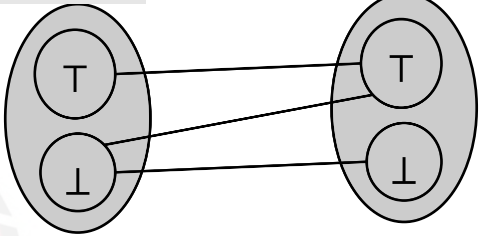
Model Based Diagnosis

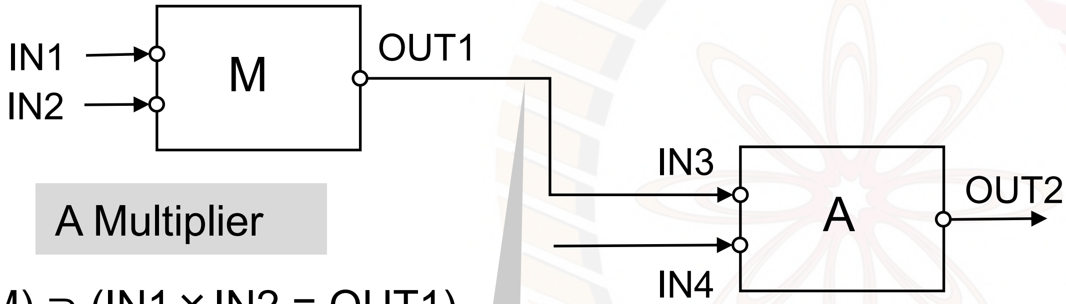


A generic approach to building model based diagnosis systems

Component models

$ok(M)$ $\supset$ $IN1 \times IN2 = OUT1$





A Multiplier

$ok(M) \supset (IN1 \times IN2 = OUT1)$

$\neg ok(M) \vee (IN1 \times IN2 = OUT1)$

An Adder

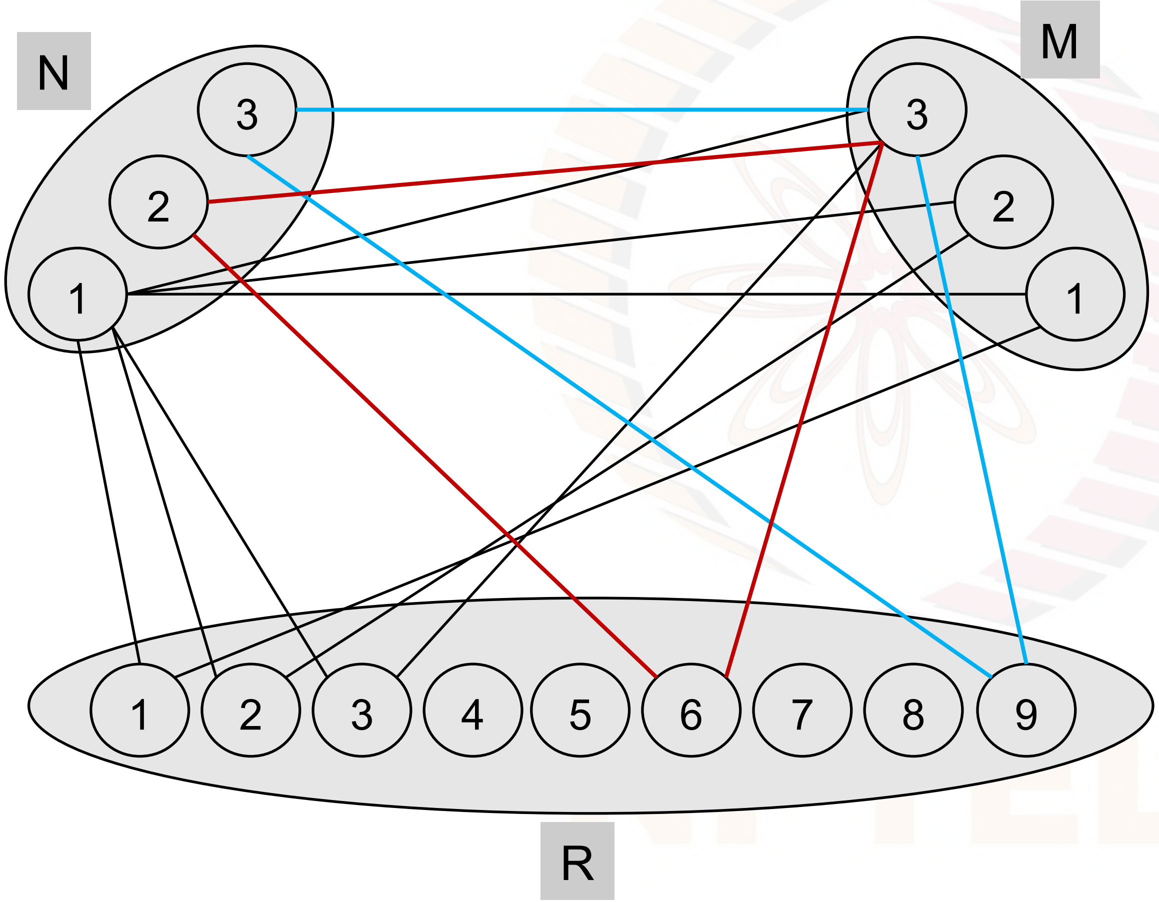
$ok(A) \supset (IN3 + IN4 = OUT2)$

$\neg ok(A) \vee (IN3 + IN4 = OUT2)$

A Connector

$OUT1 = IN3$

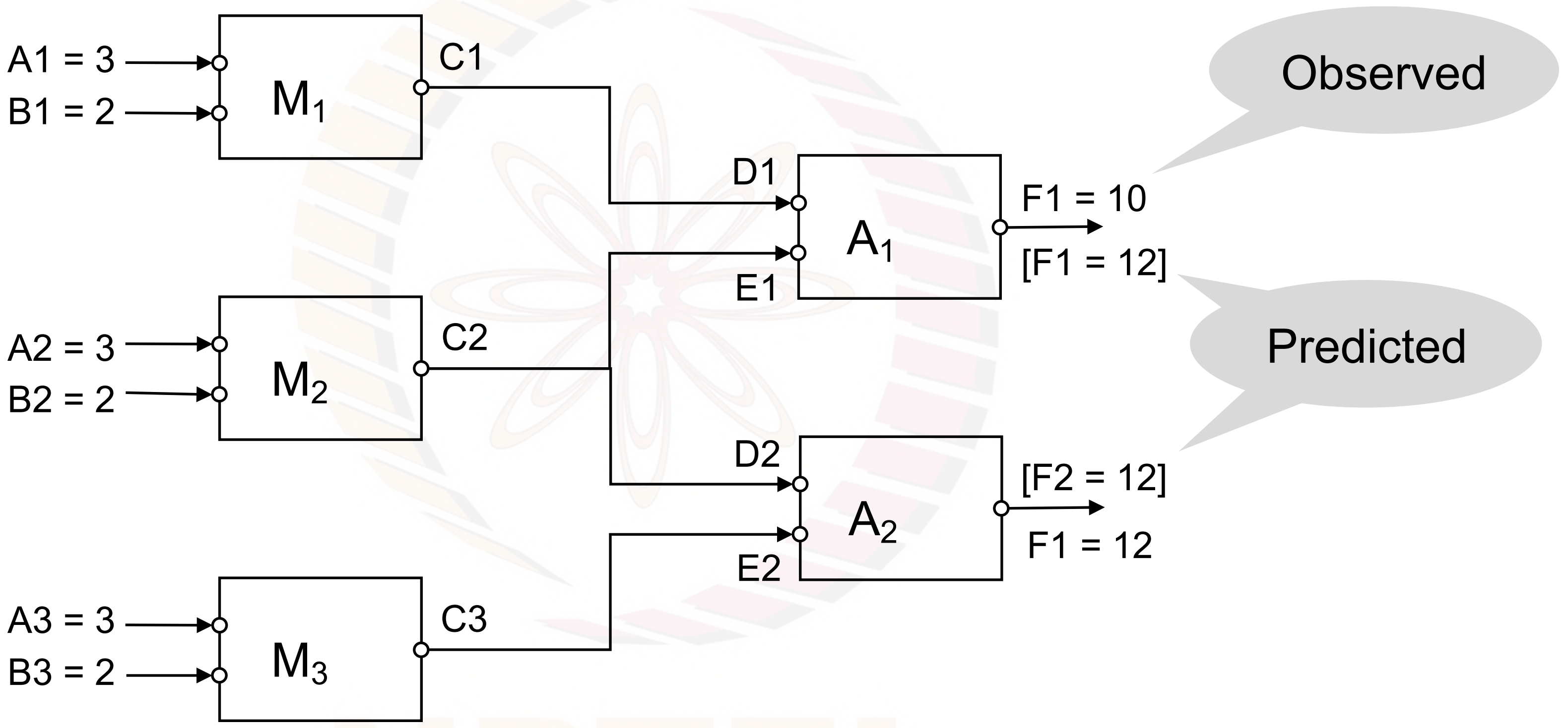
Multiplication: Ternary Constraints



$N \times M = R$

Any three values that form a triangle constitute a solution

A simple malfunctioning device



A simple device made of three multipliers and two adders

A working device: consistent constraints

$ok(M_1) \supset (A_1 \times B_1 = C_1)$

$A_1 = 3$

$B_1 = 2$

$ok(M_2) \supset (A_2 \times B_2 = C_2)$

$A_2 = 3$

$B_2 = 2$

$ok(M_3) \supset (A_3 \times B_3 = C_3)$

$A_3 = 3$

$B_3 = 2$

$C_1 = D_1$

$C_2 = E_1$

$C_2 = D_2$

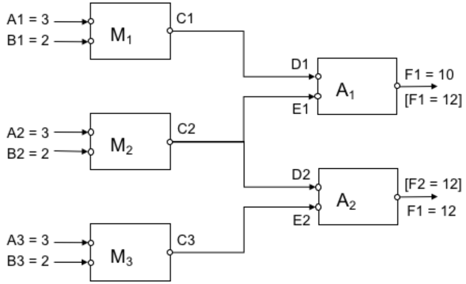
$C_3 = E_2$

$ok(A_1) \supset (D_1 \times E_1 = F_1)$

$F_1 = 12$

$ok(A_2) \supset (D_2 \times E_2 = F_2)$

$F_2 = 12$



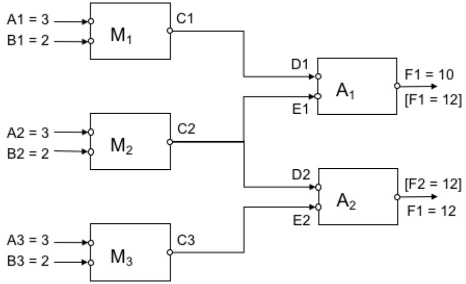
Solve the CSP →
All components ok

3 Multipliers

4 Connectors

2 Adders

A broken device: inconsistent constraints



$ok(M_1) \supset (A_1 \times B_1 = C_1)$

$A_1 = 3$

$B_1 = 2$

$ok(M_2) \supset (A_2 \times B_2 = C_2)$

$A_2 = 3$

$B_2 = 2$

$ok(M_3) \supset (A_3 \times B_3 = C_3)$

$A_3 = 3$

$B_3 = 2$

$C_1 = D_1$

$C_2 = E_1$

$C_2 = D_2$

$C_3 = E_2$

$ok(A_1) \supset (D_1 \times E_1 = F_1)$

$F_1 = 10$

$ok(A_2) \supset (D_2 \times E_2 = F_2)$

$F_2 = 12$

Some component not ok
Fault detection

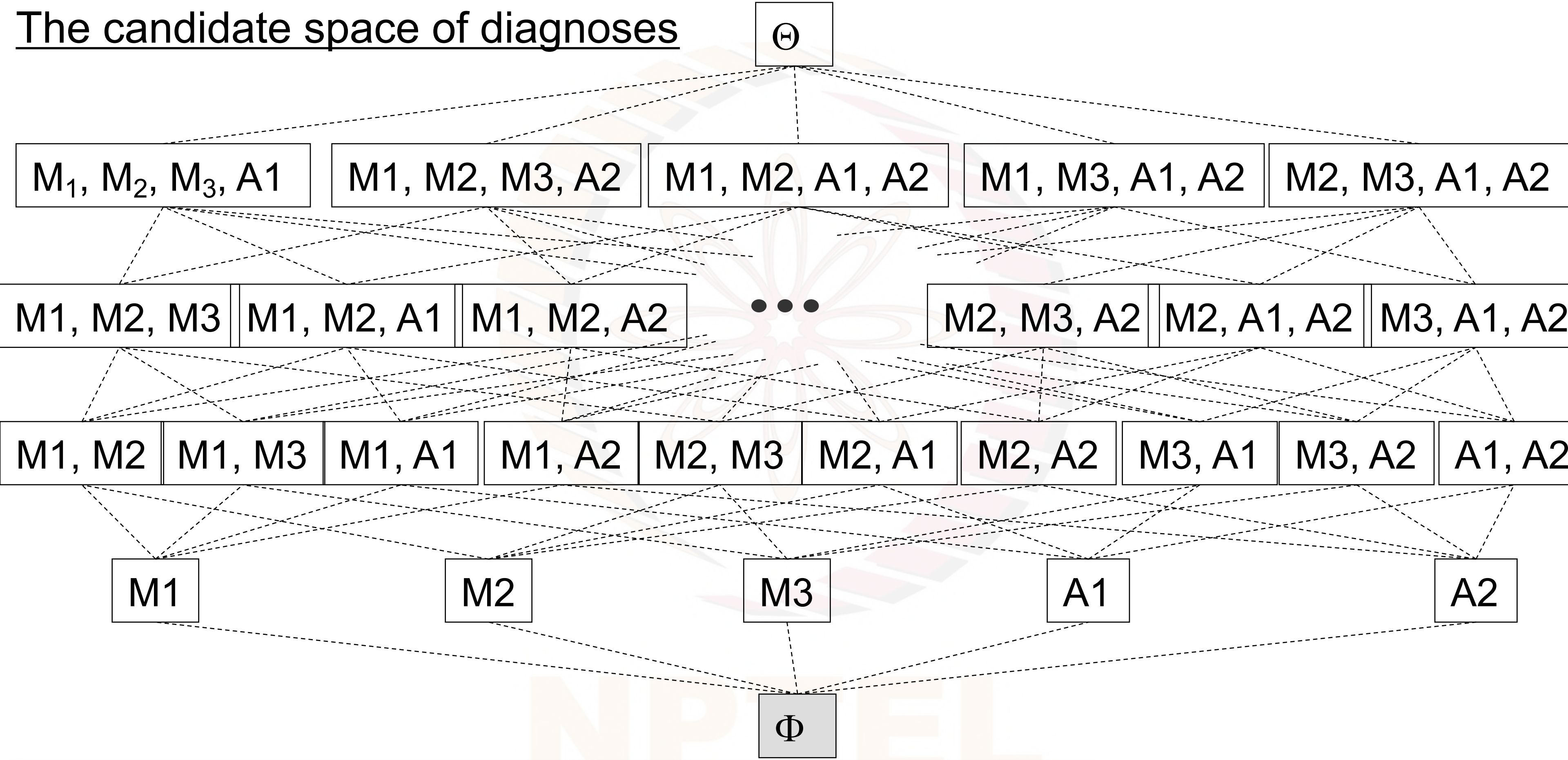
Which component?
Fault localization
Solve CSP → solution

3 Multipliers

4 Connectors

2 Adders

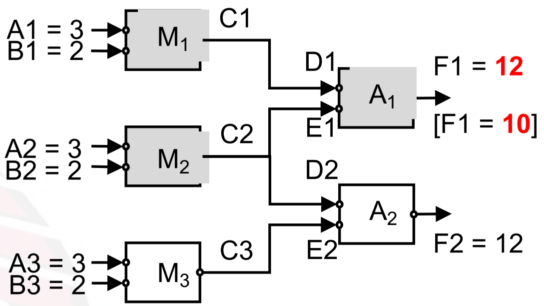
The candidate space of diagnoses



The *minimal candidate* when the device is working is the empty set.

Inconsistency leads to conflict sets

Note: ok(component) ≡ ¬Ab(component)



The predicted value *F1*=12 is based on the *assumptions* ¬Ab(M_1), ¬Ab(M_2) and ¬Ab(A_1)

The value *F1*=10 is an observation, and is based on no assumption.

The cumulative assumptions

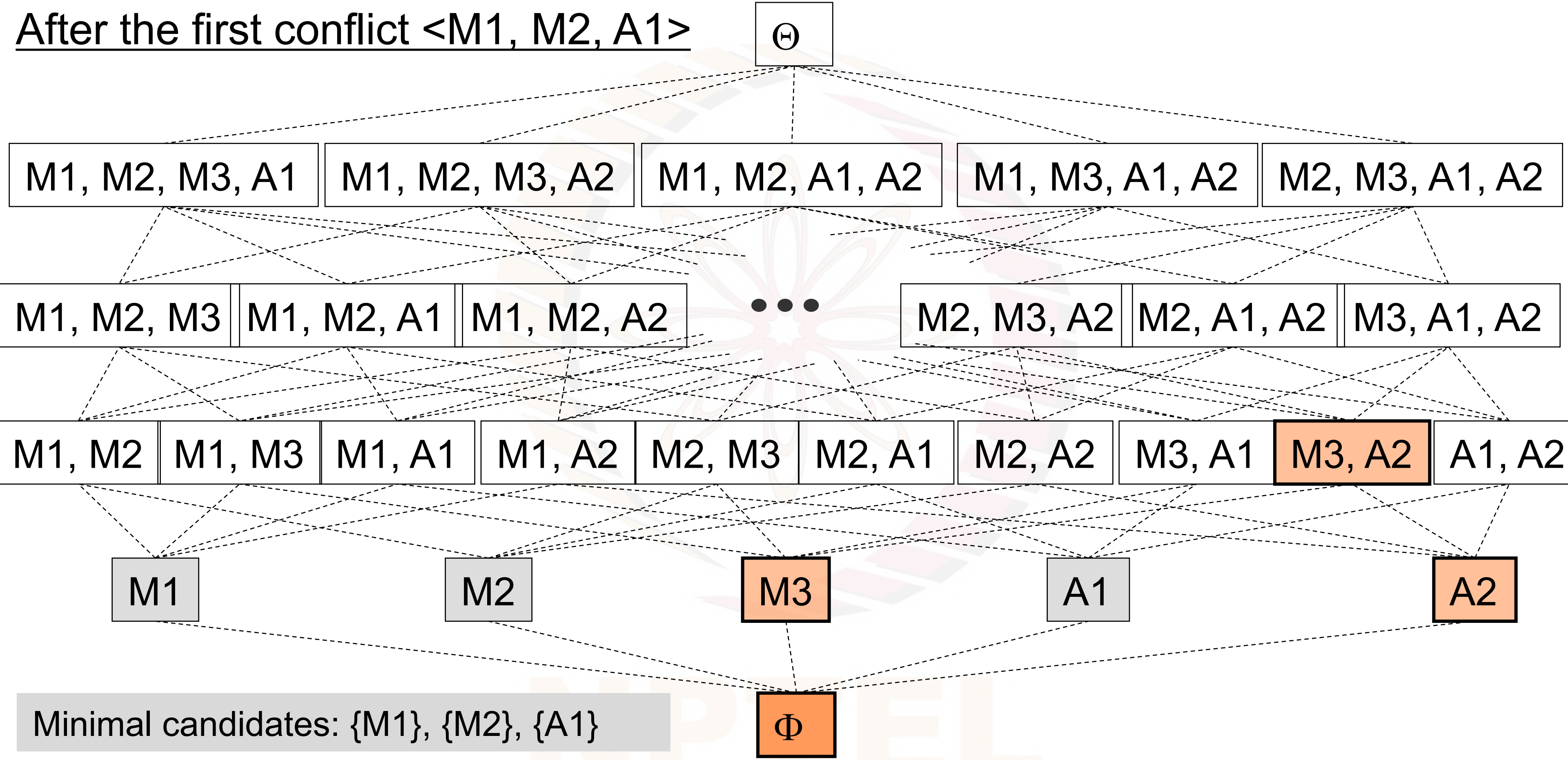
from the two ways of arriving at the value are inconsistent together.

That is, ¬Ab(M_1), ¬Ab(M_2) and ¬Ab(A_1) cannot be true at the same time.

We represent this as the conflict $<M_1, M_2, A_1>$

Can M_3 or A_2 be broken along with M_2? → $<A_1, A_2, M_1, M_3>$

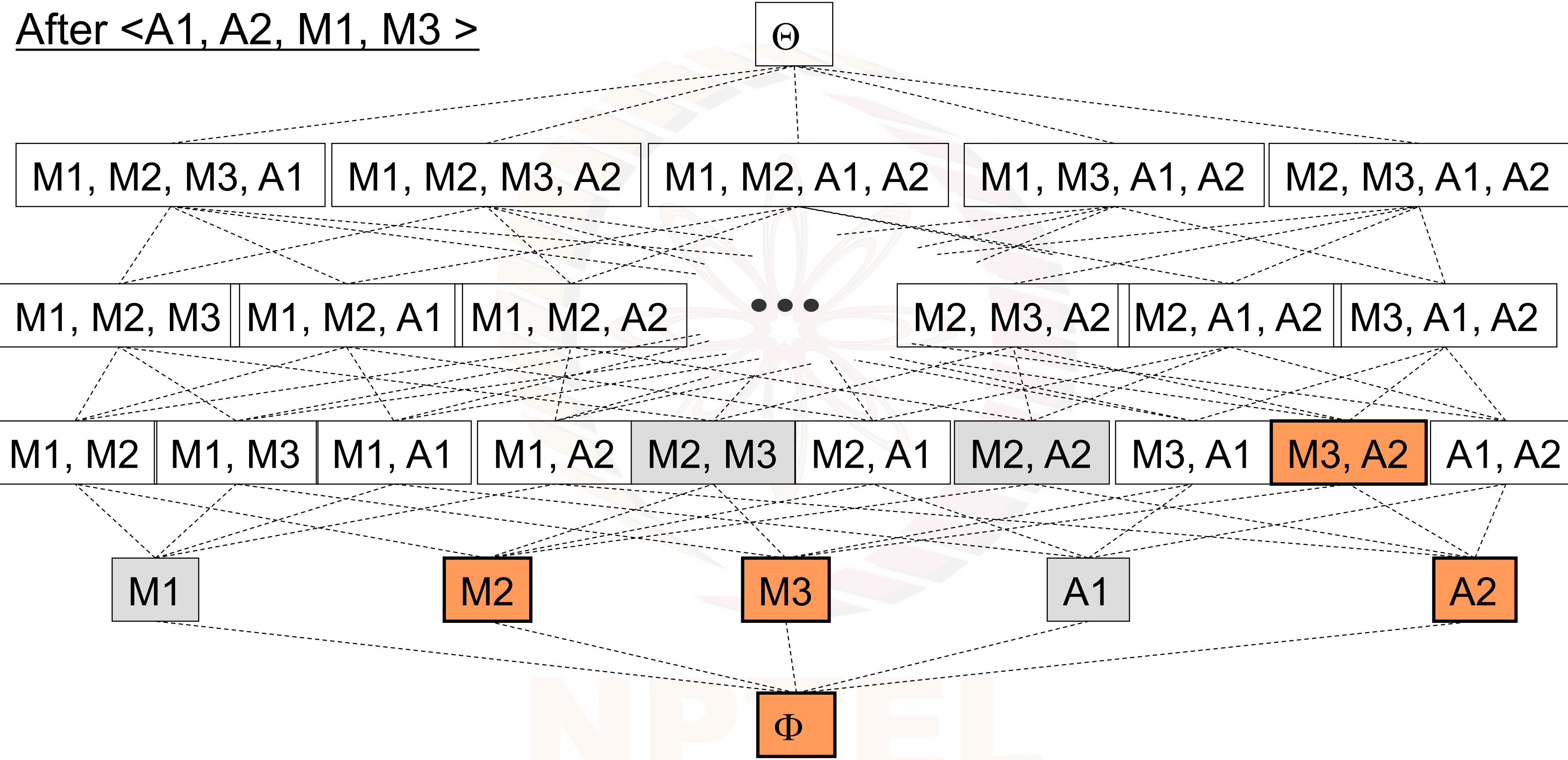
After the first conflict <M1, M2, A1>



Minimal candidates: {M1}, {M2}, {A1}

{Φ}, {M3}, {A2} and {M3, A2)} eliminated

After <A1, A2, M1, M3 >

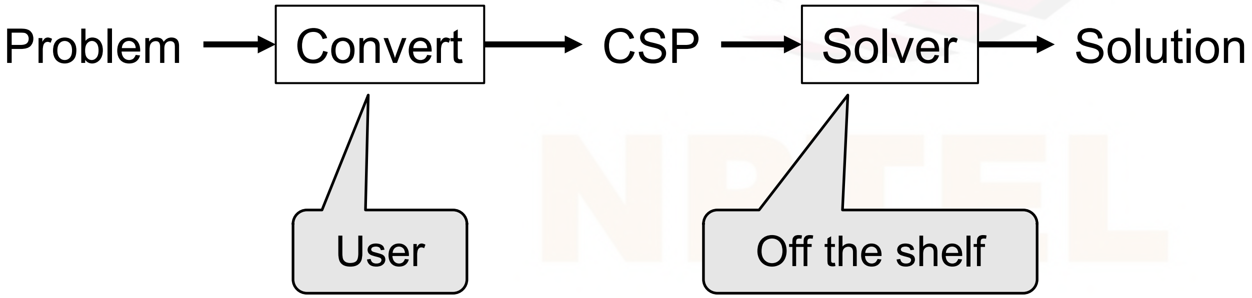


Constraint Processing: Posing and Solving CSPs

Problems that can be posed as CSPs

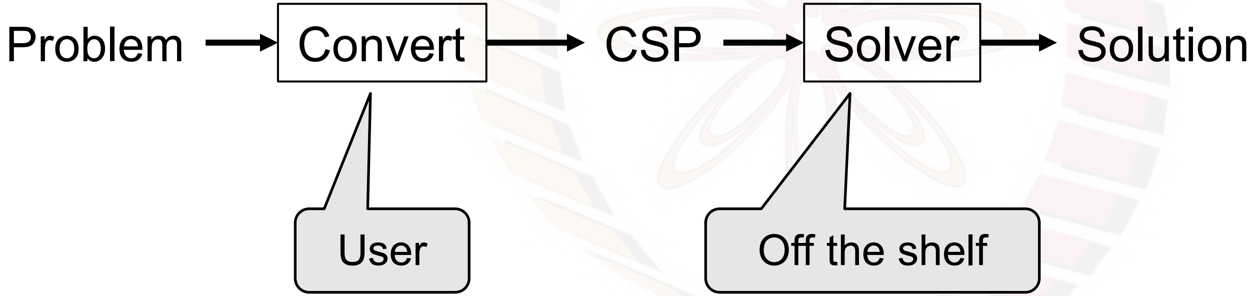
- SAT: a special case of CSP where each domain = {true, false} / {1, 0}
- Map Colouring: naturally posed as a CSP
- Planning: Planning Graph = CSP, SATPLAN (Kautz and Selman, 1996)
- Consistency Based Diagnosis
- Scheduling

... and many others



Next: CSP solvers

Constraint Processing: Solving CSPs



Some (informal) terminology

An assignment A_Z

assigns values to a subset Z of variables, $Z \subseteq X$

Let $\mathcal{A} = \langle a_Z, a_{Z-1}, \ldots, a_1 \rangle^*$ be the tuple of values in A

Let $\mathcal{A}_S$ be the *projection* of $\mathcal{A}$ on the set of variables S

Then A_Z *satisfies* a constraint C = (S, R)

where S is the *scope* of R

if $S \subseteq Z$ and $\mathcal{A}_S \in R$

An assignment A_Z is *consistent*

if for every constraint C = (S, R) s.t. $S \subseteq Z$

A_Z satisfies C

A *solution* is a consistent assign for all the variables in X

* order to cater to the algorithm which adds new values at the head

A search algorithm for solving a CSP

Let $X = (x_1, x_2, ..., x_N)$ be the order in which the N variables are tried

Let $D_i = (a_{i1}, a_{i2}, a_{i3}, ...)$ be the values in domain D_i in the order they will be tried

The search algorithm, *Backtracking*, is as follows

For each variable x_i

Try the values in its domain one by one till

a value *consistent* with earlier variables is found

If a *consistent* value is found then advance to the next variable x_{i+1}

else go back to x_{i-1} and try the next *untried* value

Termination happens in two ways

1. All values for x_1 are exhausted without a solution
2. The last variable x_N is assigned a consistent value

Algorithm Backtracking

```
BACKTRACKING (X, D, C)
  𝒜 ← [ ]
  i ← 1
  D'_i ← D_i
  while 1 ≤ i ≤ N
      a_i ← SELECTVALUE(D'_i, 𝒜, C)
      if a_i = null
          then    i ← i − 1
                  𝒜 ← tail 𝒜
          else    𝒜 ← a_i : 𝒜
                  i ← i + 1
                  if i ≤ N
                      then D'_i ← D_i
  return REVERSE(𝒜)
```

Initializing

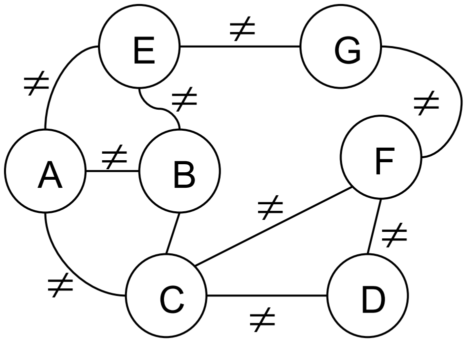
Copy the domain

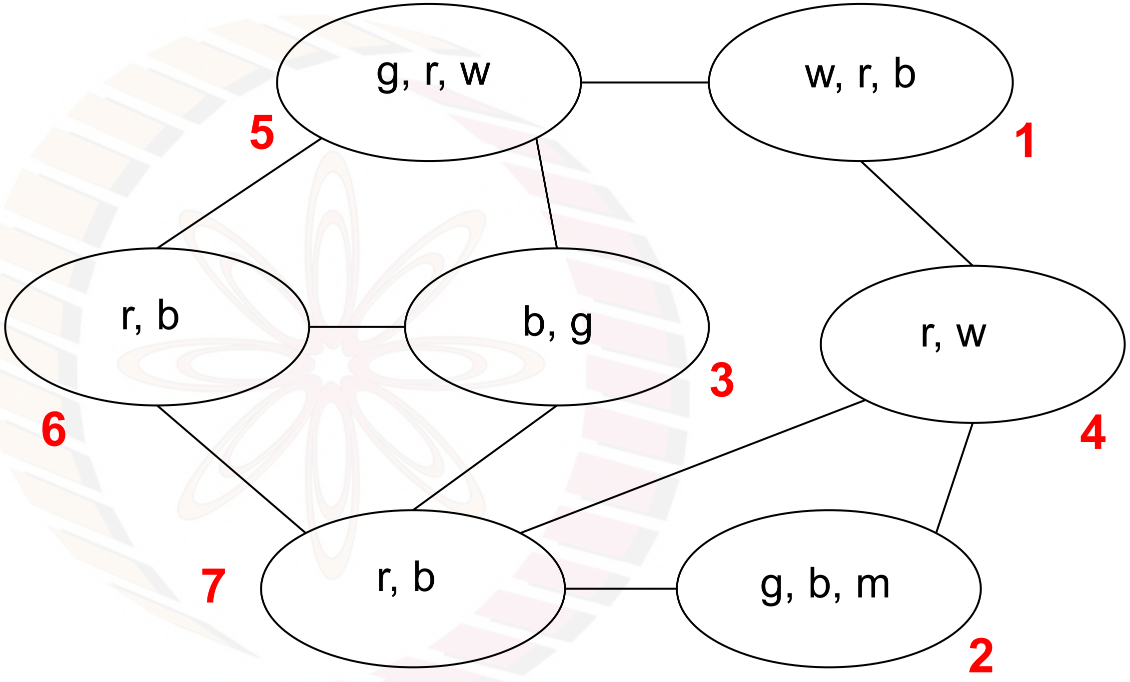
Backtracking

Augmenting

```
SELECTVALUE(D'_i, 𝒜, C)
  while D'_i is not empty
      a_i ← head D'_i
      D'_i ← tail D'_i
      if CONSISTENT(a_i : 𝒜)
          then return a_i
  return null
```

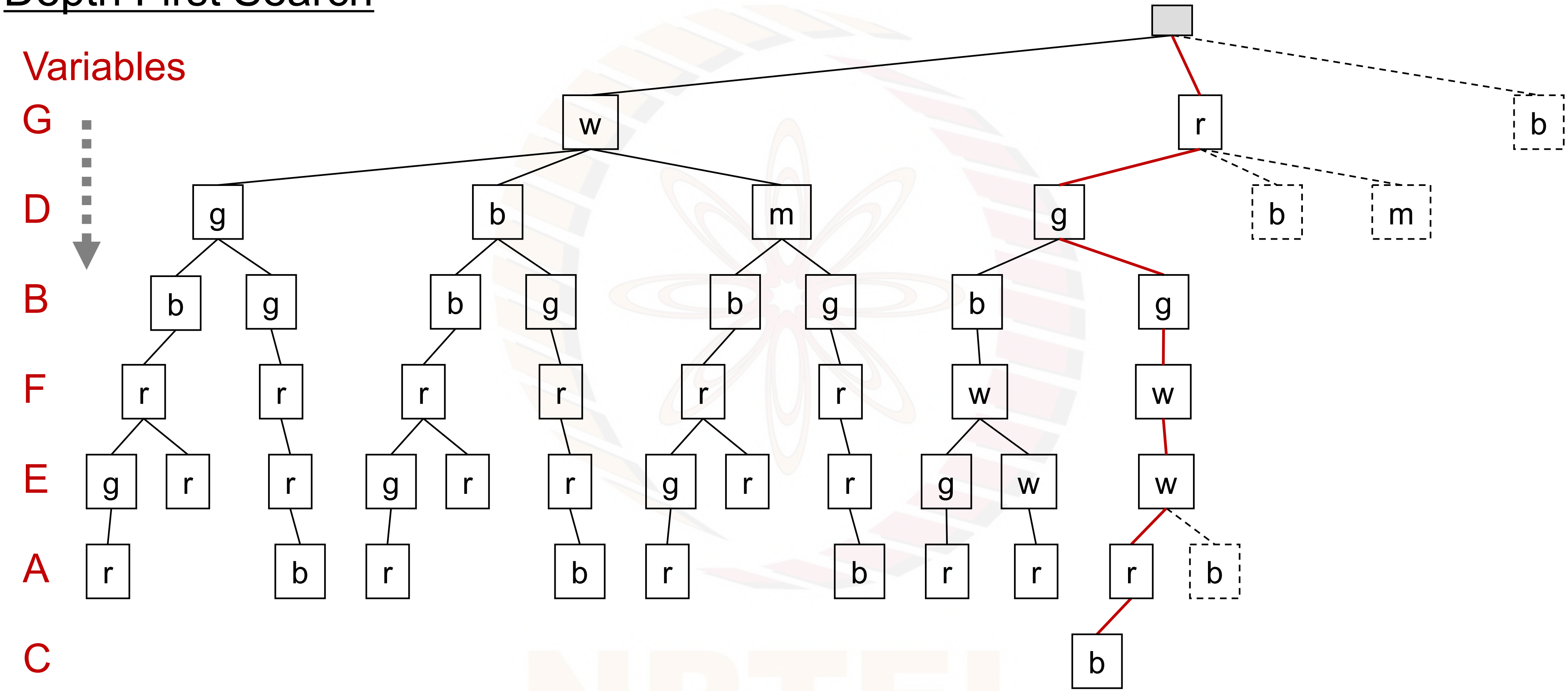
A map colouring problem





The fixed ordering chosen is GDBFEAC
depicted by numbers on the right.

Depth First Search



The tree explored by *Backtracking* when it finds the first solution

Battling CombEx

There are various approaches to combat combinatorial explosion

- Choosing an appropriate ordering of nodes
  - Min-induced-width ordering of the constraint graph
  - Select nodes with higher degree first
- Dynamic Variable Ordering
  - Choose variables with smallest domains first
- Preprocess the network $\mathcal{R}$ to prune the search space
  - Consistency enforcement

Coming up

- Prune the domains during search
  - Lookahead Search

Coming up

- Intelligent Backtracking
  - Lookback Search
  - *Memoization*: remember *nogoods*

Arc Consistency

- Let (X,Y) be an edge in the constraint graph of a network $\mathcal{R}$
- Variable X is said to be arc-consistent w.r.t variable Y iff
  for every value $a \in D_X$ there a variable $b \in D_Y$ s.t. $\langle a,b \rangle \in R_{XY}$
- X can be made arc-consistent w.r.t Y by algorithm Revise

```
REVISE((X), Y))
1.   for every a ∈ D_X
2.      if there is no b ∈ D_Y s.t. <a,b> ∈ R_XY
3.         then delete a from D_X
```

Complexity: $\mathcal{O}(k^2)$

- An edge (X,Y) is said to be arc-consistent if both X and Y are arc-consistent w.r.t. each other
  - achieved by calls to Revise((X),Y) and Revise(Y)), X)

- A network is arc-consistent if all its edges are arc-consistent

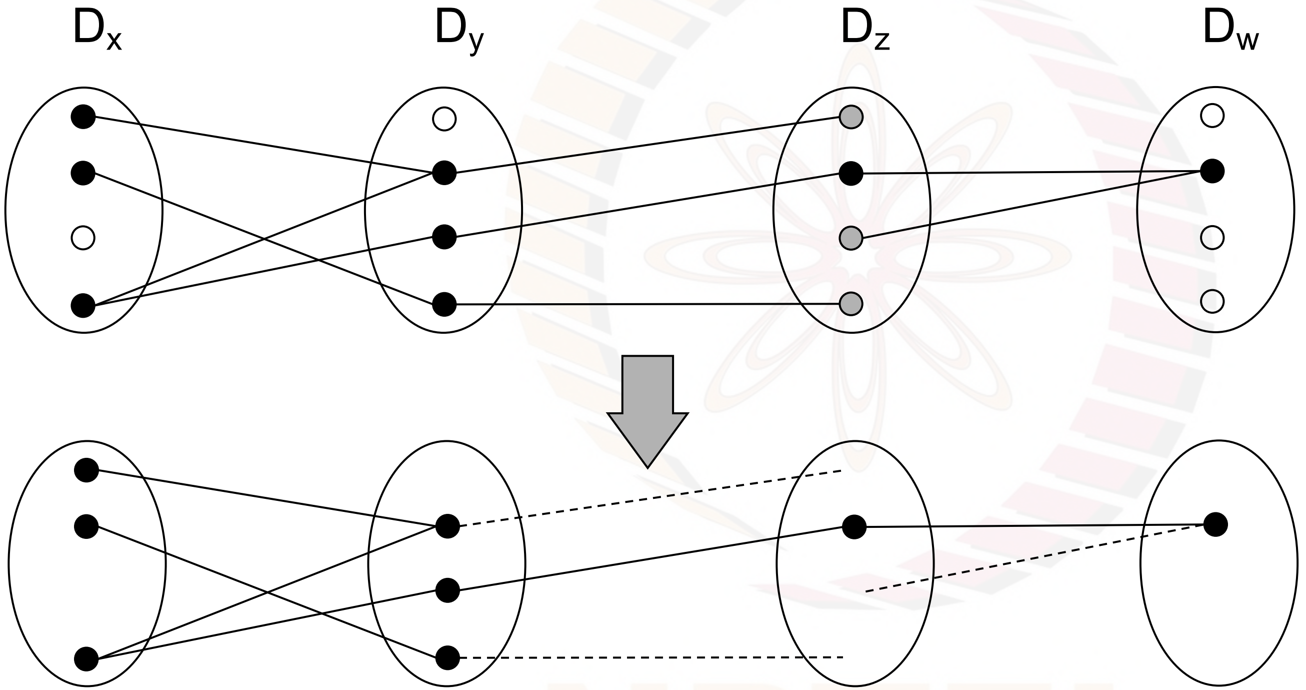
Arc Consistent Networks

- A network is arc-consistent if all its edges are arc-consistent
- Will a cycle of Revise calls over all edges,
  in both directions do the trick?

For each edge (X,Y) in the constraint graph
    Call Revise((X),Y)
    Call Revise((Y),X)

- The effect Revise((X) Y) is to prune the domain D_X
- Can a disappearing value
  from the domain of a variable at the end of an edge
  affect the arc consistency of another edge?
- The following example shows us that the answer is yes

One cycle of calls to Revise is not enough



After Revise with ((x),y), ((y),x), ((y),z), ((z), y), ((z),w) and ((w),z)
As one can see two values in D_y are unsupported at this stage

Algorithm AC-1

The algorithm AC1 cycles through all edges as long as even one domain changes

```
AC-1 (X, D, C)
1.  repeat
2.      for each edge (X,Y) in the constraint graph
3.          REVISE((X), Y))
4.          REVISE((Y), X))
5.   until no domain changes in the cycle
```

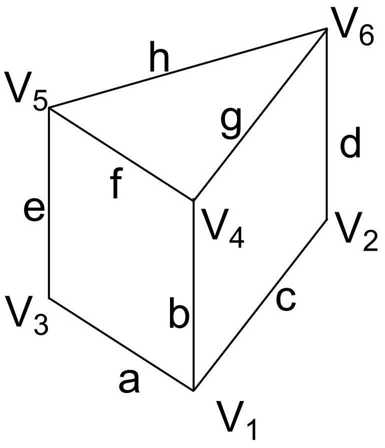
Complexity: $\mathcal{O}(nek^3)$

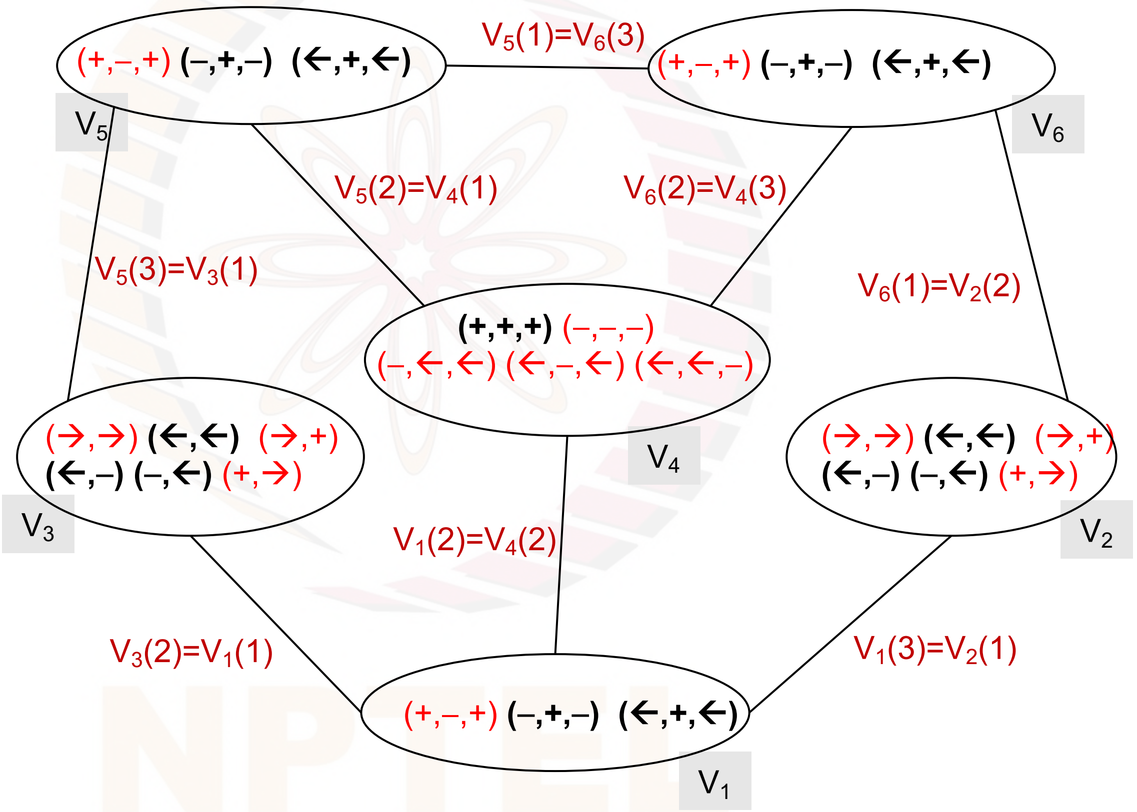
Let there be *n* variables, each with domain of size *k*
Let there be *e* edges in the constraint graph
Every cycle has complexity $\mathcal{O}(ek^2)$

In the worst case the network is not arc-consistent and
in every cycle exactly one element in one domain is removed,

So there are *nk* cycles

An arc-consistent version

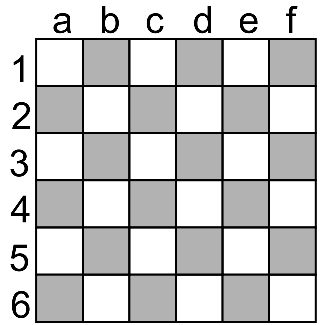




1 2 3

W vertex: V_1= (a,b,c)
L vertex: V_2= (c,d)
L vertex: V_3= (e,a)
Y vertex: V_4= (b,f,g)
W vertex: V_5= (h,f,e)
W vertex: V_6= (d,g,h)

6-Queens: Binary Constraint Network



Variables: one variable for each of the 36 squares

Domains: {Q, nil}

Constraints: one binary constraint {R_{XY}}

R_{XY} = {<a1=Q,a2=0>, <a2=Q,a1=0>, ..., <f6=Q,a1=0>}

constraint – pair of locations where two queens *cannot* be placed

Variables: {a, b, c, d, e, f, g} columns

Domains: {1, 2, 3, 4, 5, 6} rows

Constraints: {R_{ab}, R_{ac}, ..., R_{fg}} pairs of columns

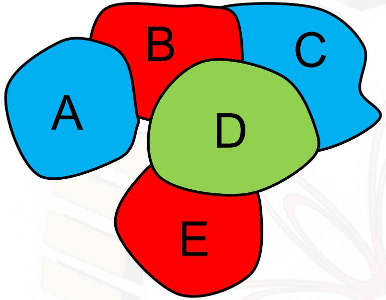
R_{XY}: *Allowed* rows of queens in columns X and Y

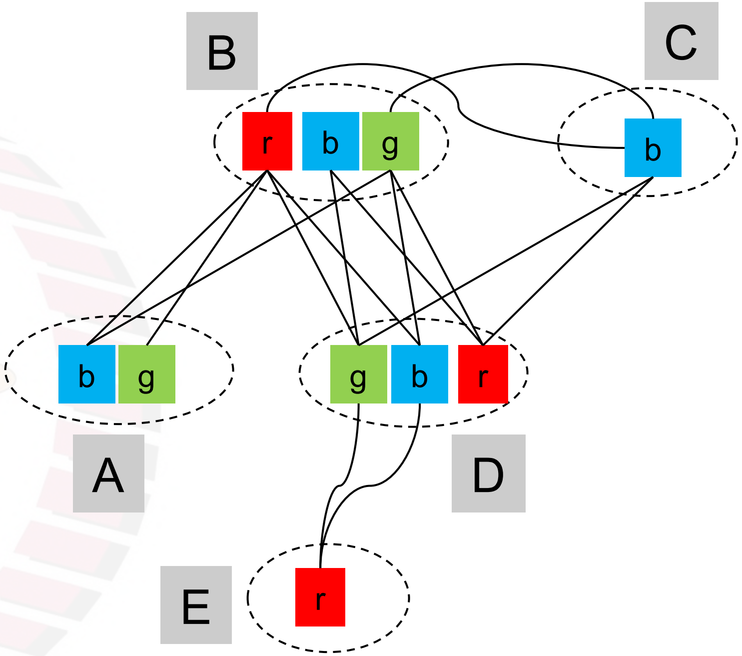
R_{ab} = {<1,3>, {<1,4>, <1,5>, <1,6>, <2,4>, ... ,<4,6>}

Q: Are the above networks arc-consistent?

Map Colouring *(recap)*

Find a solution





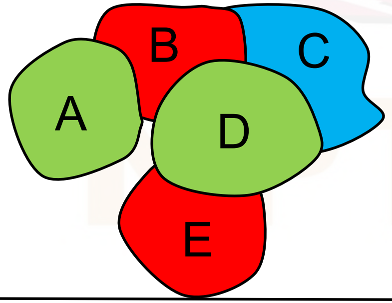
Is there a colouring in which B = b?

No

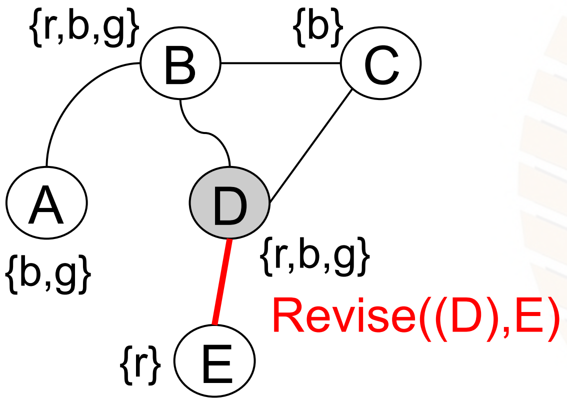
Is there a colouring in which B = g?

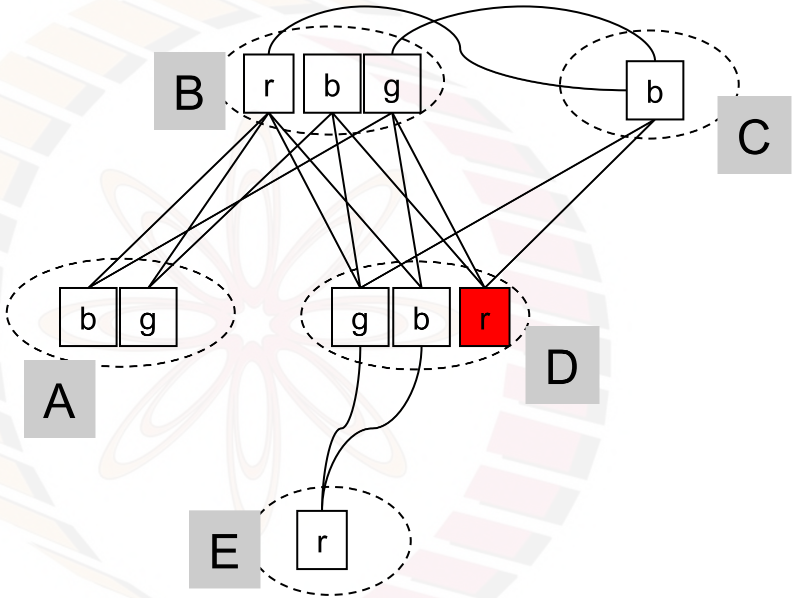
No

Is there a colouring in which A = g?



Arc Consistency



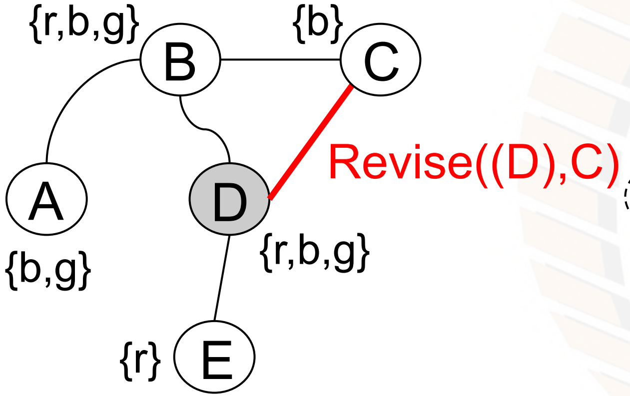


The constraint graph

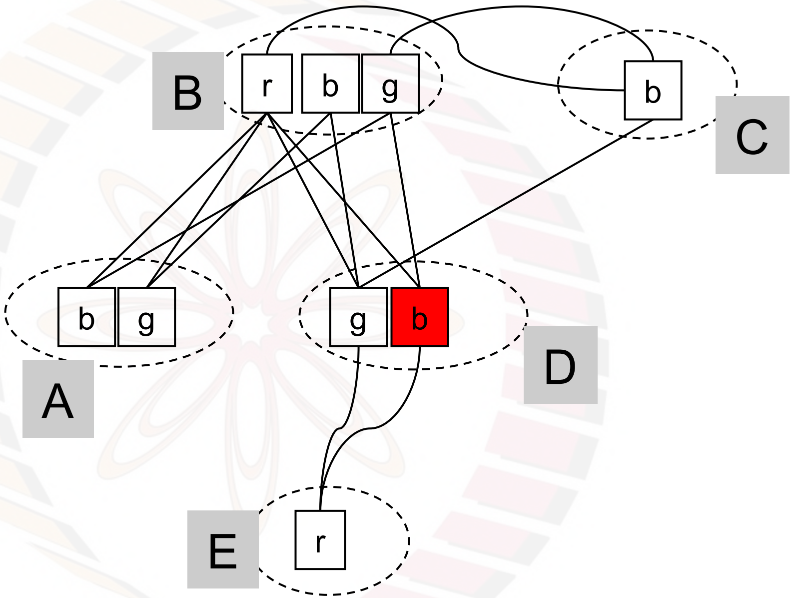
The matching diagram

value ‘r’ is removed because there is no corresponding value in E

Arc Consistency



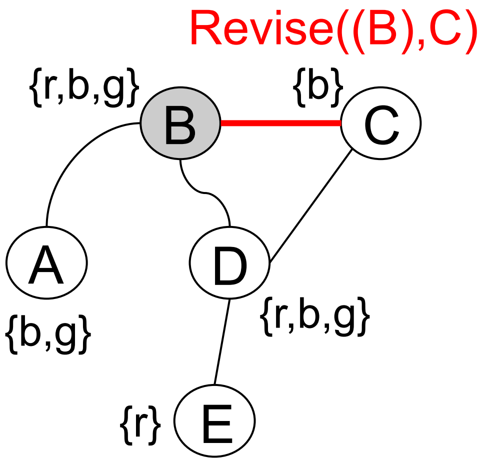
The constraint graph



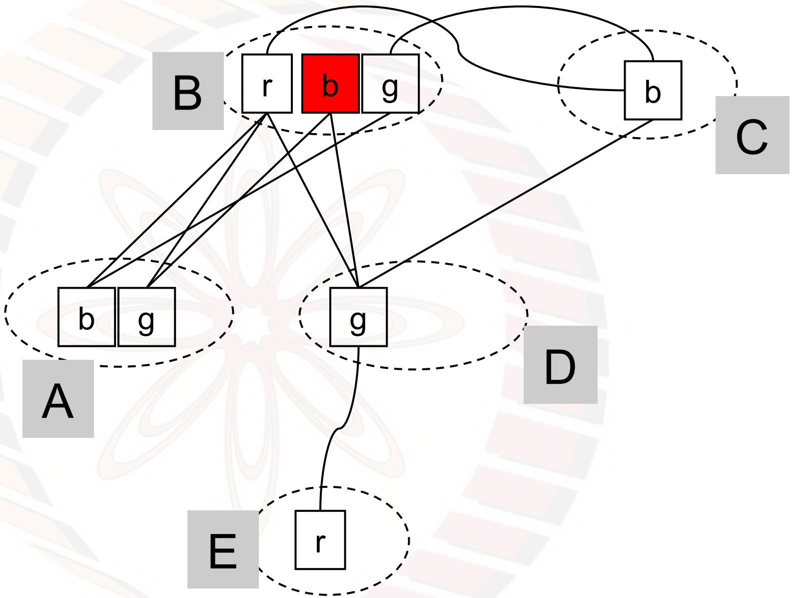
The matching diagram

Value ‘b’ is removed because there is no corresponding value in C

Arc Consistency



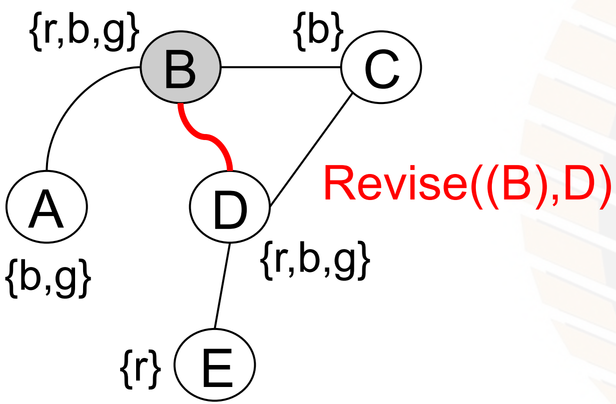
The constraint graph



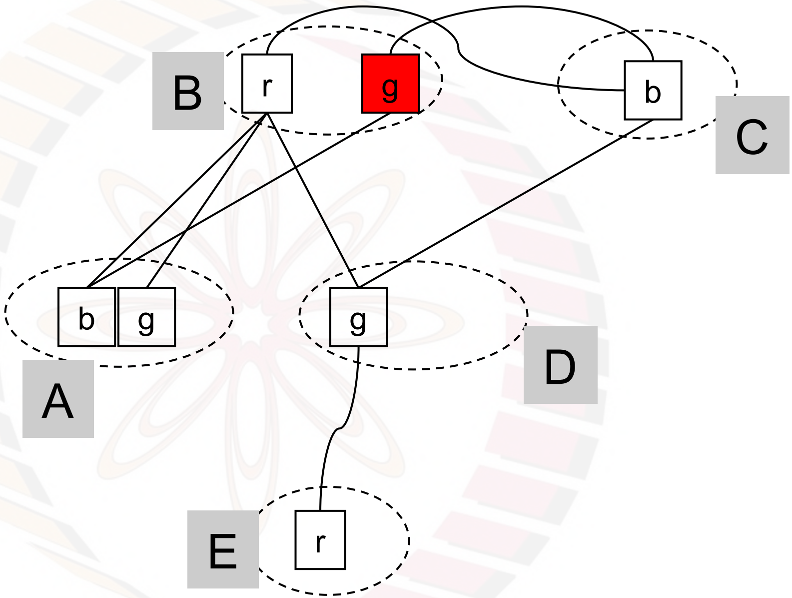
The matching diagram

Value ‘b’ is removed because there is no corresponding value in C

Arc Consistency



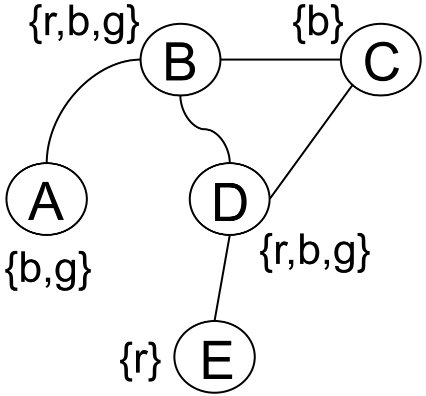
The constraint graph



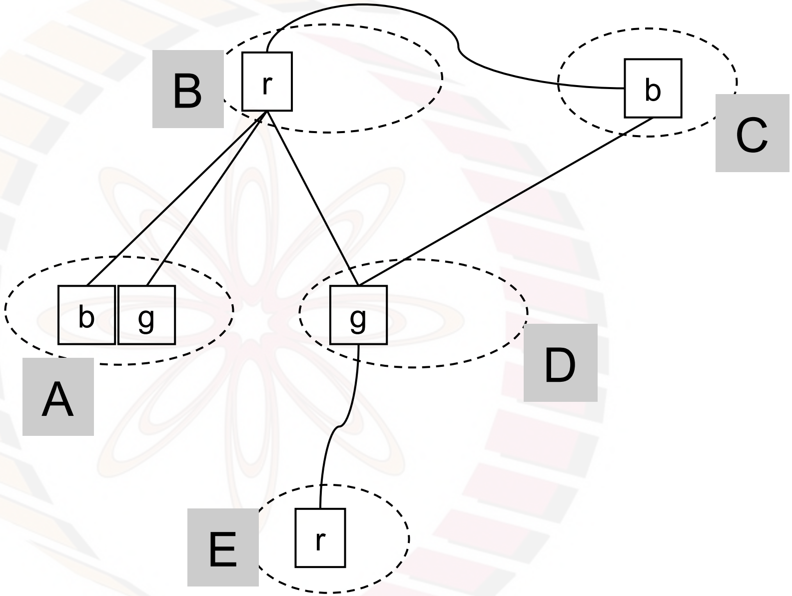
The matching diagram

Value ‘g’ is removed because there is no corresponding value in D

Arc Consistent



The constraint graph

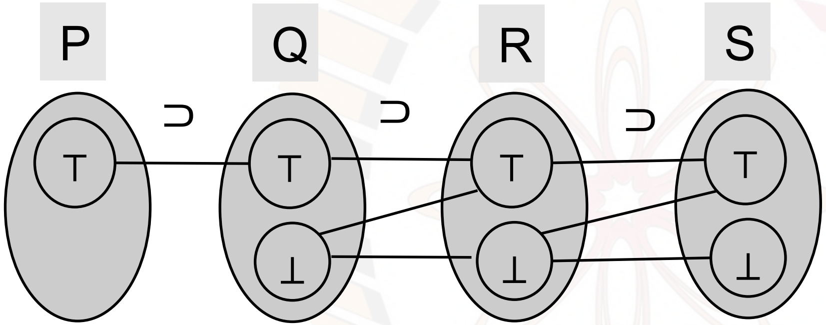


The matching diagram

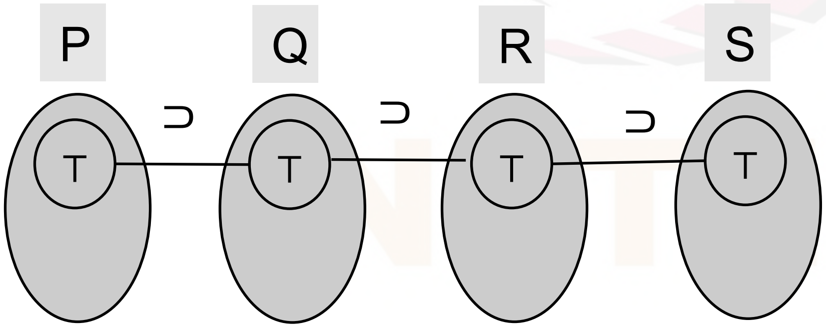
The network is now arc consistent
For *some* networks arc-consistency results in backtrack-free search

Consistency Enforcement = Reasoning

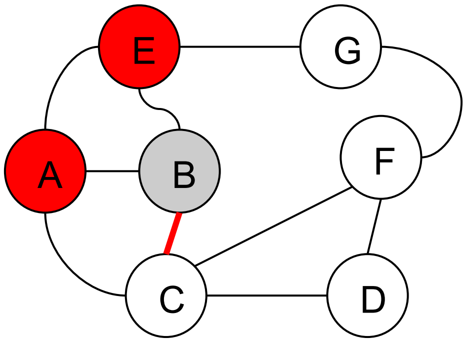
The knowledge base {P, P⊃Q, Q⊃R, R⊃S} corresponds to the following CSP



Enforcing arc-consistency



Propagation



In the constraint graph shown here
let the domain of B change after Revise((B), C)

An element *b* deleted from D_B could be the *only* support
for some elements in the domains of A and E

This means that
the edges (E,B) and (A,B) could no longer be arc-consistent.

Therefore one must evoke Revise((A), B) and Revise((E), B) again.

This is the essence of algorithm AC-3.

A change in a variable is *propagated* to the connected variables.
Only those are considered again for consistency

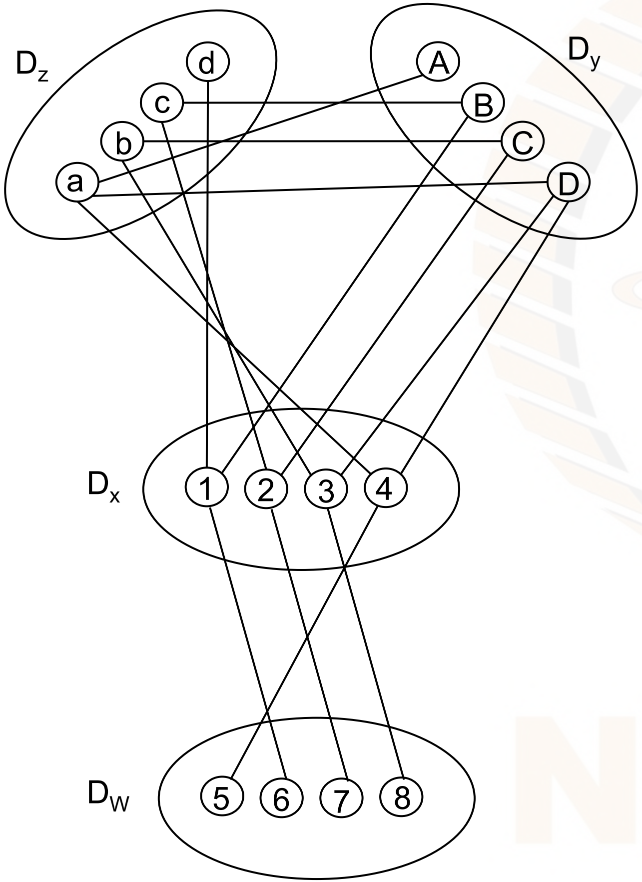
Algorithm AC-3

```
AC-3 (X, D, C)
    Q ← [ ]
    for each edge (N,M) in the constraint graph
            Q ← Q ++ (N,M) : [(M,N)]
    while Q is not empty
        (P,T) ← head Q
        Q ← tail Q
            REVISE((P), T))
            if D_P has changed
                for each R ≠ T and (R,P) in the constraint graph
                        Q ← Q ++ [(R,P)]
```

Complexity: $\mathcal{O}(ek^3)$

If the domain of a variable P has changed
then consistency w.r.t P is enforced for the neighbours of P

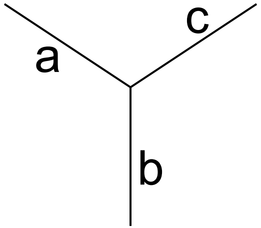
How many calls in AC-3?

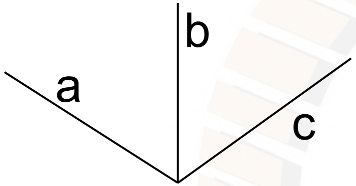


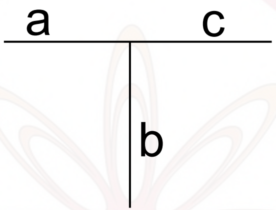
Simulate AC-1 and also AC-3 on the given matching diagram after initializing the queue with (x,y), (y,x), (y,z), (z,y), (z,w) and (w,z)

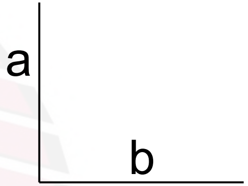
Q: How many calls to Revise are made in AC-1?
Q: How many calls to Revise are made in AC-3?

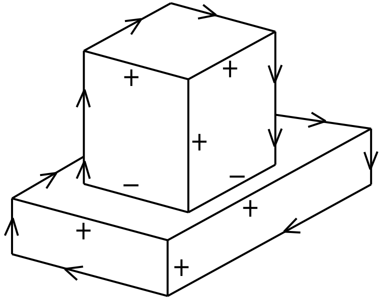
Interpreting line drawings of trihedral objects











Y joint or Fork (a,b,c)

- (+,+,+)
- (–,–,–)
- (–,←,←)
- (←,–,←)
- (←,←,–)

W joint or Arrow (a,b,c)

- (+,–,+)
- (–,+,–)
- (←,+,←)

T joint (a,b,c)

- (←,←,←)
- (←,→,←)
- (←,+,←)
- (←,–,←)
- (→,+,→)
- (→,–,→)

L joint (a,b)

- (→,→)
- (←,←)
- (→,+)
- (←,–)
- (–,←)
- (+,→)

Waltz Algorithm

- David Waltz extended the domain defined by Huffman
  - more than three-edge vertices
  - objects with cracks
  - images with light and shadows
- The number of edge labels shot up to 50+
- The number of valid vertices shot up to thousands
- The Waltz Algorithm is somewhere between AC-1 and AC-3
- It does propagation from vertex to vertex
- The video with the link below shows the algorithm in action.
- It removes the lines depicting shadows and cracks, and produces a drawing with only object edges.

A video "David Waltz - Constraint Propagation" on YouTube

i-Consistency

- A network is said to be *i*-consistent if an assignment to any (i-1) variables can be consistently extended to *i* variables.
- 1-consistency = node consistency
  - for example specifying that a Boolean variable P to a particular value
- 2-consistency = arc consistency
- 3-consistency = path consistency
  - any edge in the matching diagram can be extended to a triangle
-
-

The higher the level of consistency the lower the amount of backtracking that the search algorithm does

Lookahead Search

Achieving i-consistency *before* embarking upon search
results in a smaller search space being explored.

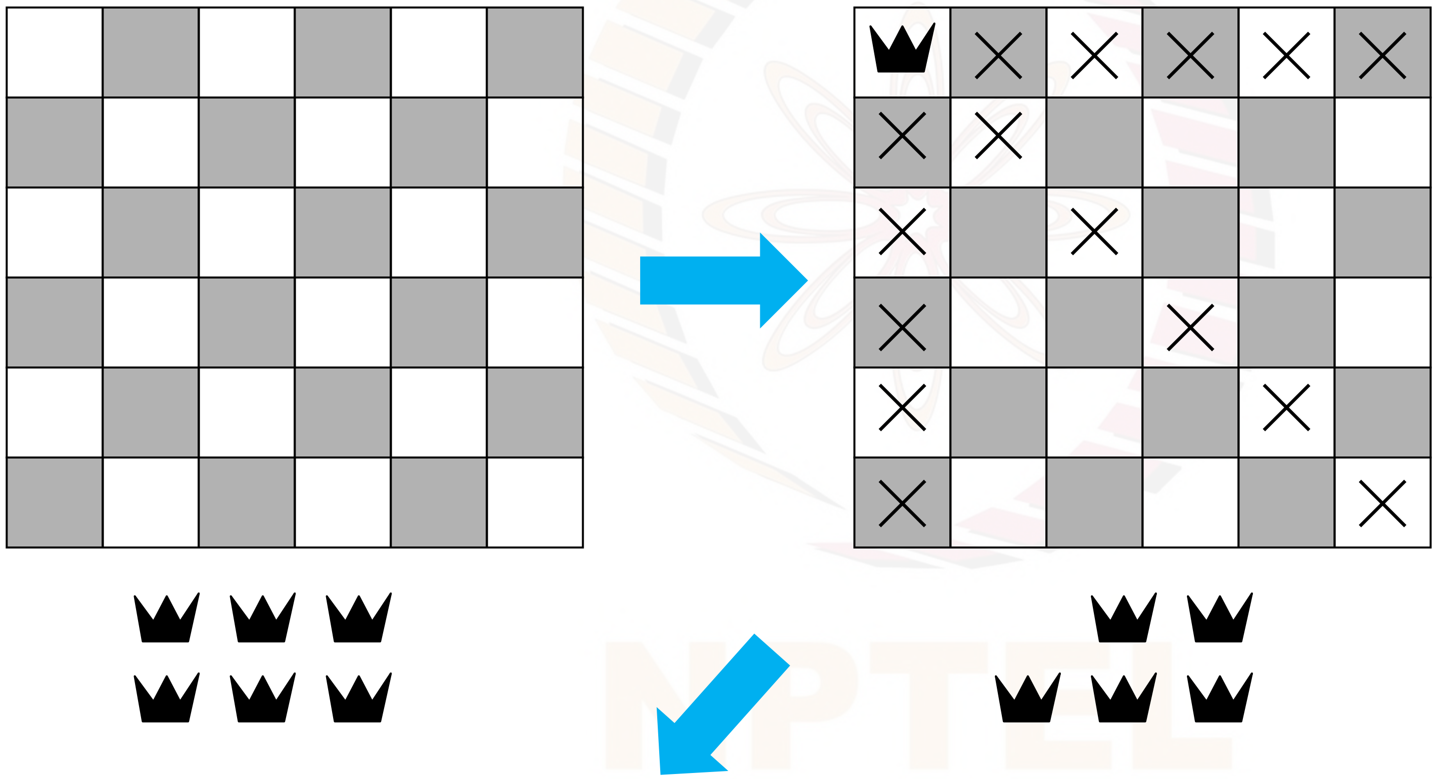
Algorithm Backtracking chooses the value for the next variable
in an arbitrary order.

Can one choose the value in a more intelligent manner?

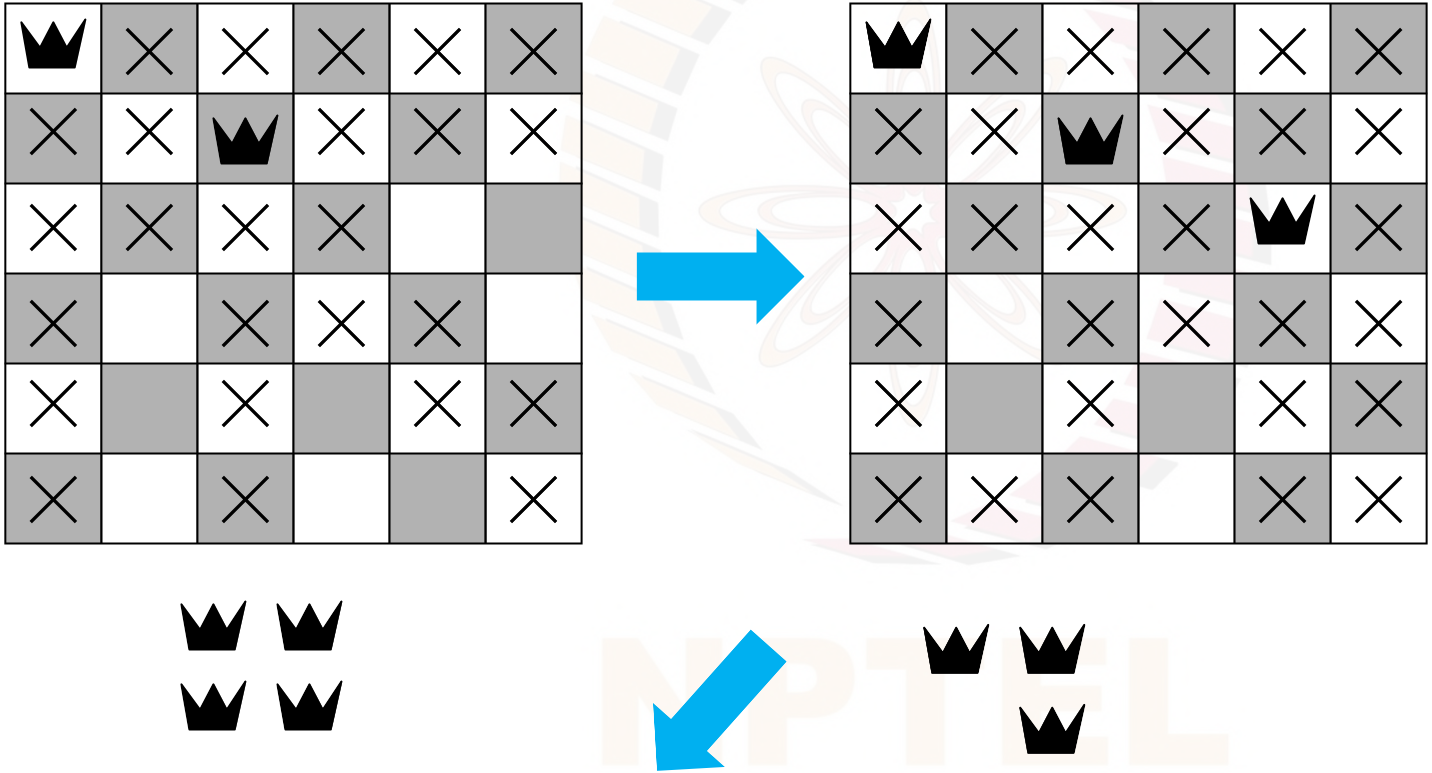
Lookahead search variations inspect all values to estimate
which one would lead to fewer conflicts in the future

We look at one – Algorithm *ForwardChecking*
It prunes future domains removing inconsistent values

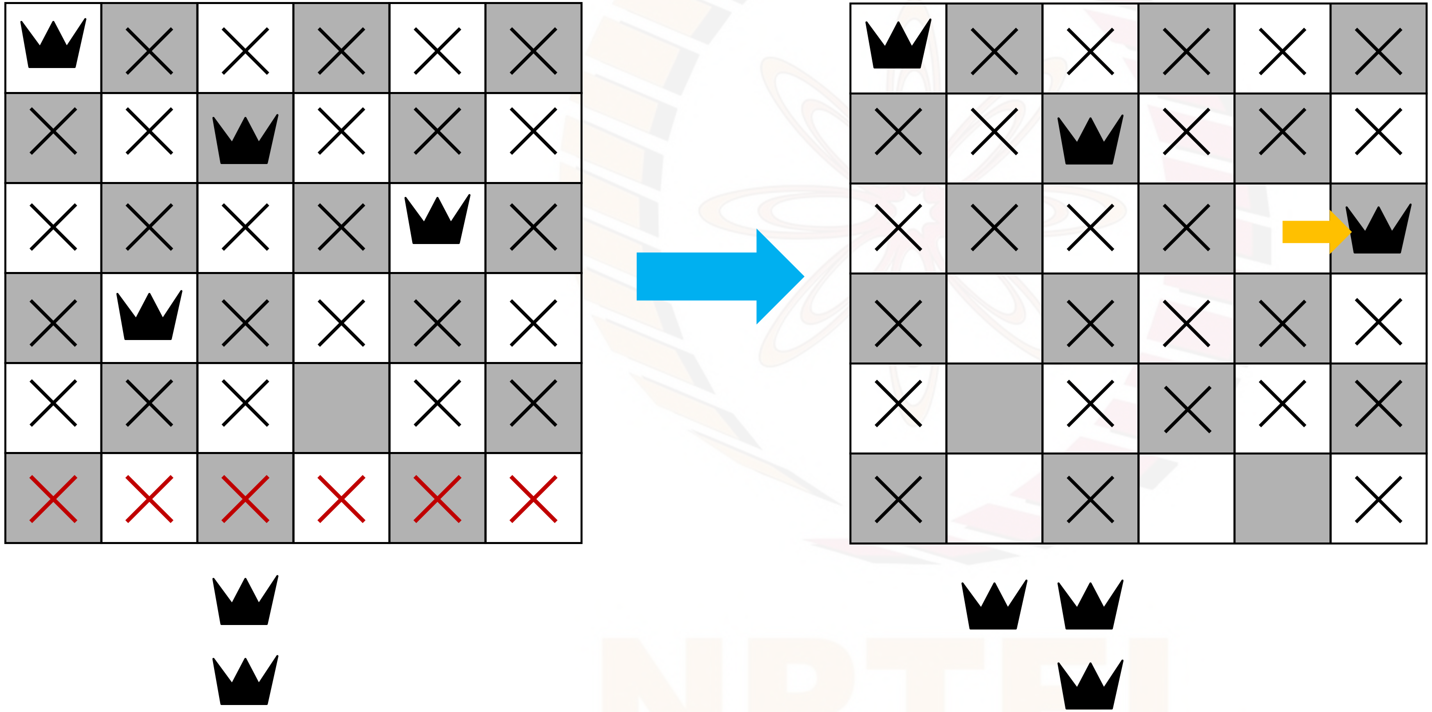
6-Queens: placing queens row wise



Rows 2 and 3



Row 4: dead-end



FC backtracks, but Queen 4 will again result in dead-end

Lookback Search

- Algorithm Backtracking does *chronological* backtracking
- This means that on reaching a *dead-end*
  Backtracking looks for another value for the *previous variable*
- Can one choose the *variable* in a more intelligent manner?
- *Jumpback* methods aim to identify *culprit* variables
  - the cause of the deadend
- Lookback search variations investigate different ways
  of identifying the culprit
  - Based on graph topology
  - Based on values that cause the conflict
    - beyond the scope of *this* course

Algorithm ForwardChecking

```
FORWARDCHECKING (X, D, C)
𝒜 ← [ ]
for k ← 1 to N
    D'_k ← D_k
i ← 1
while 1 ≤ i ≤ N
    a_i ← SELECTVALUE-FC(D'_i, 𝒜, C)
    if a_i = null
        then    i ← i − 1
                𝒜 ← tail 𝒜
        else    𝒜 ← a_i : 𝒜
                i ← i + 1
                if i ≤ N
                    then D'_i ← D_i
return REVERSE(𝒜)
```

Copy all domains.
Forward Checking aims to delete values of future variables inconsistent with the value for the current variable being considered

A different function to select the current value

Algorithm SelectValue-FC

```
SELECTVALUE-FC(D'_i, 𝒜, C)
  while D'_i is not empty
        a_i ← head D'_i
        D'_i ← tail D'_i
           for k ← i +1 to N
              for each b in D'_k
                  if not CONSISTENT(b : a_i : 𝒜)
                     delete b from D'_k
           if no D'_k is empty
                then return a_i
                else for k ← i +1 to N
                     undo deletes in D'_k
  return null
```

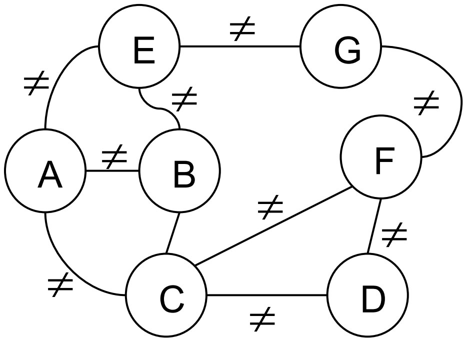
Forward Checking deletes values of future variables inconsistent with the value for the current variable being considered.

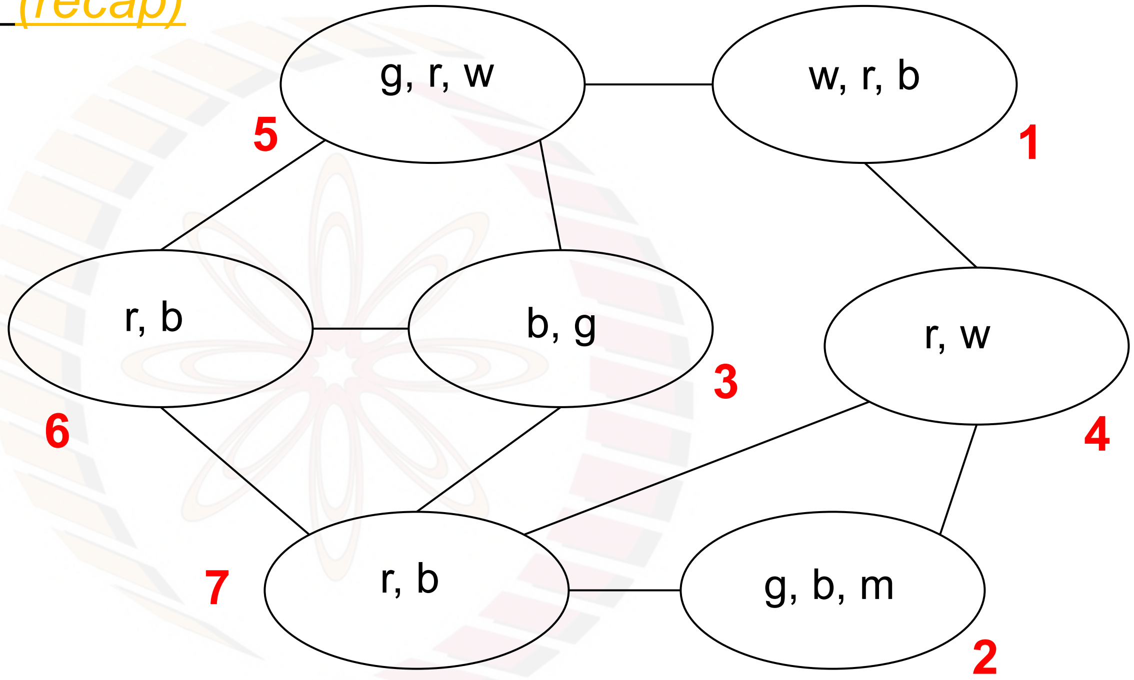
Return a_i only if all future domains are not empty

Else undo deletes done in this round

Forward Checking

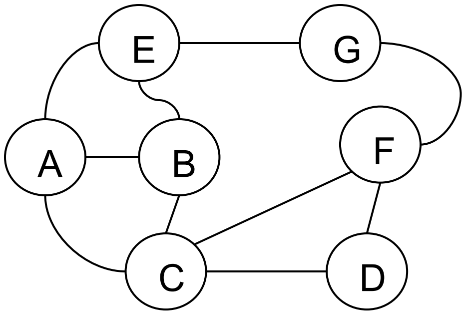
A map colouring problem *(recap)*

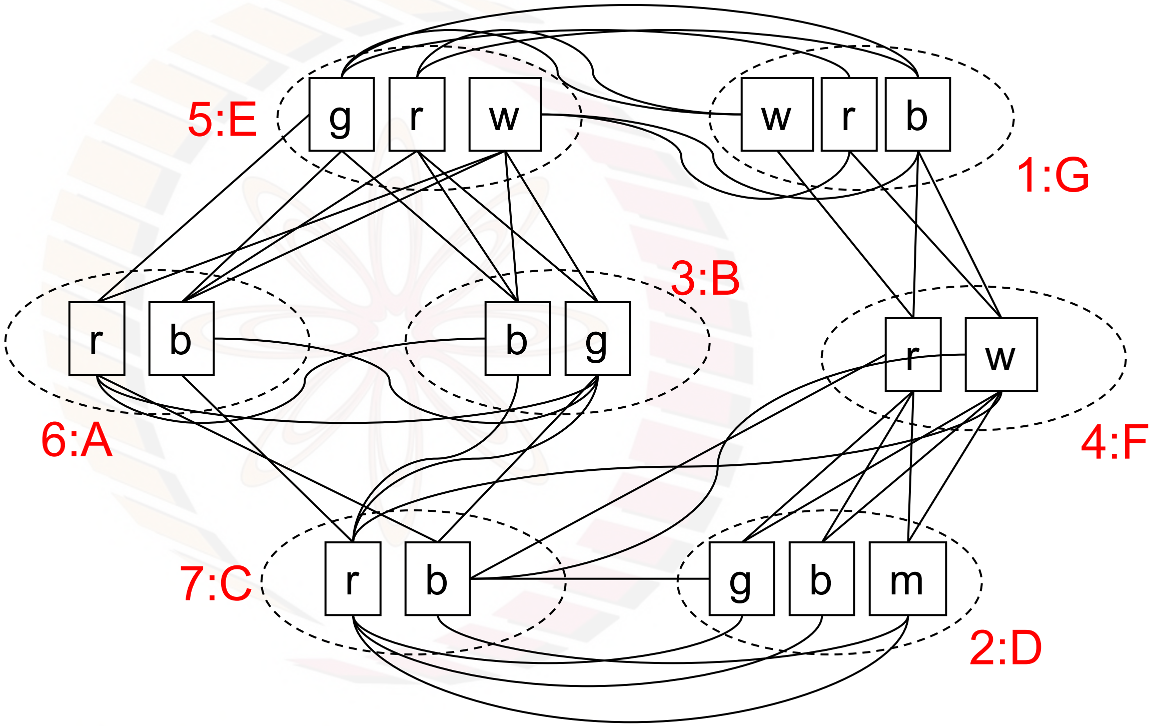


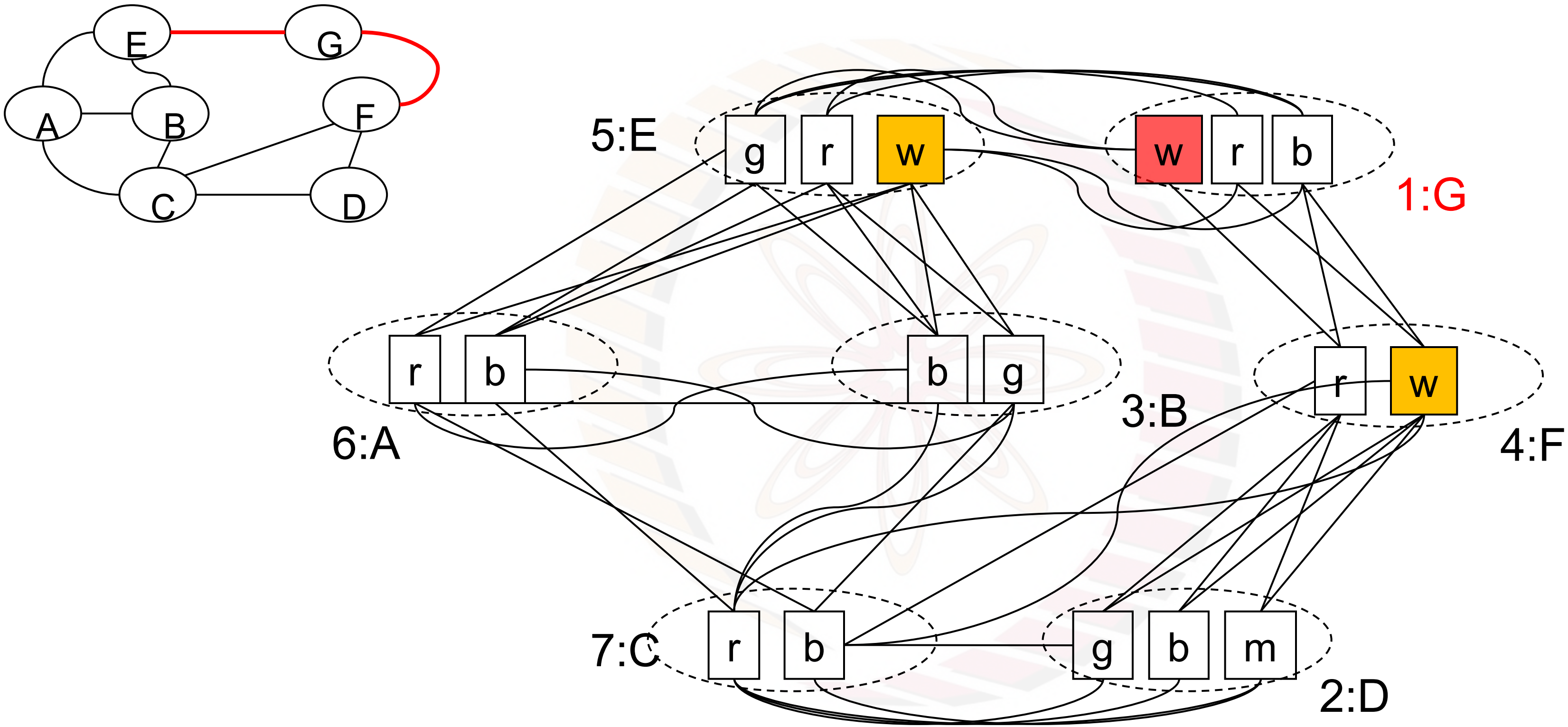


The fixed ordering chosen is GDBFEAC depicted by numbers on the right.

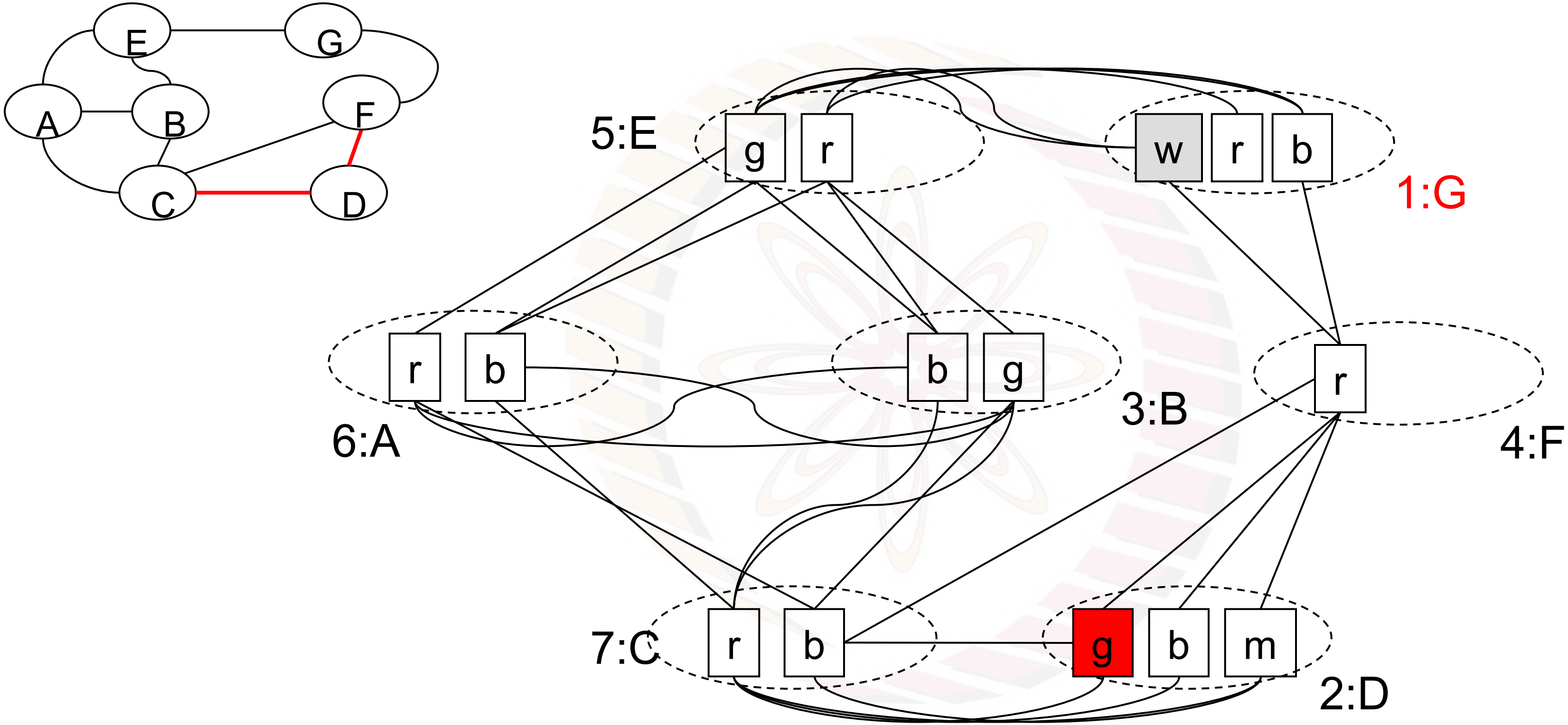
Map Colouring: The Matching Diagram





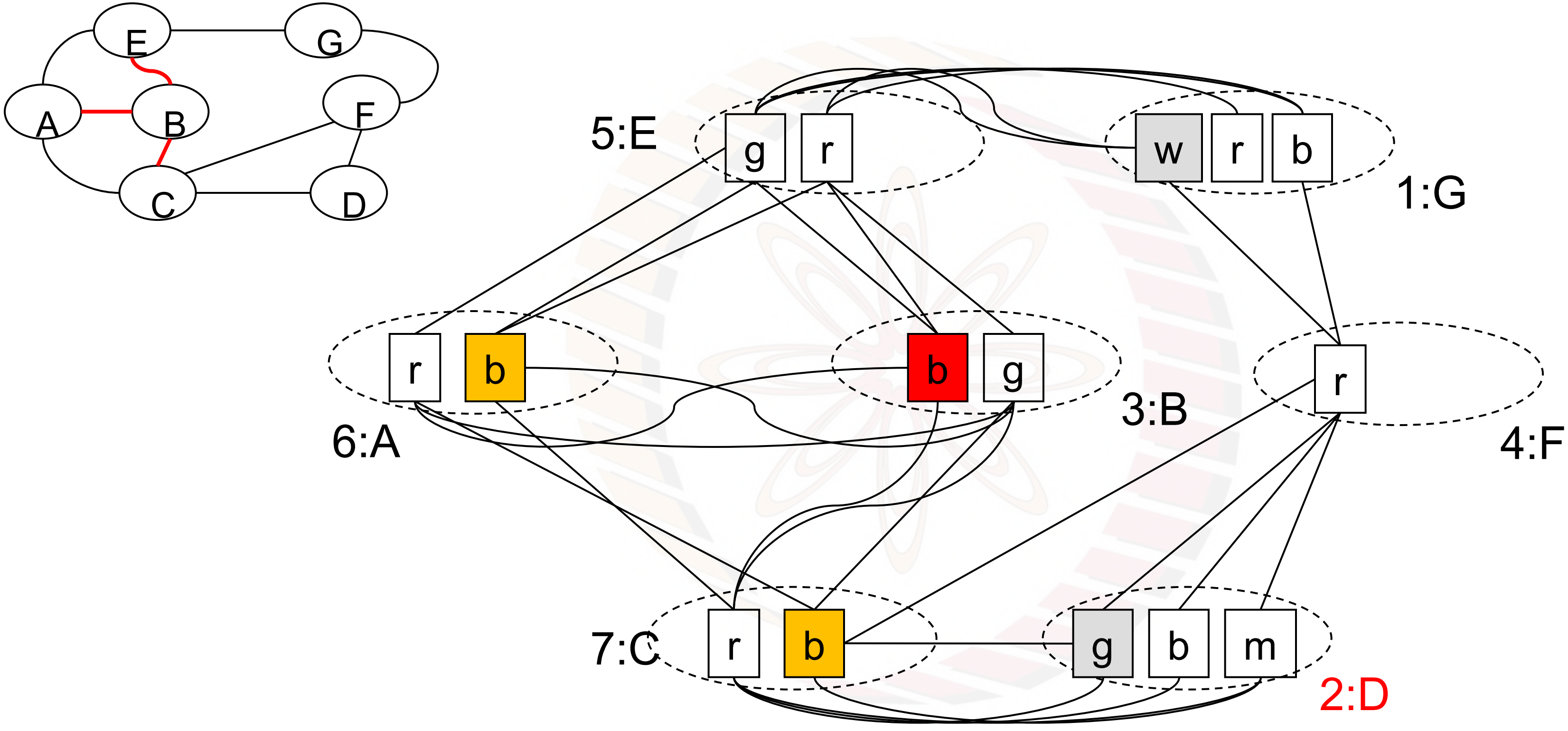


Forward Checking begins by picking value *w* for *G*

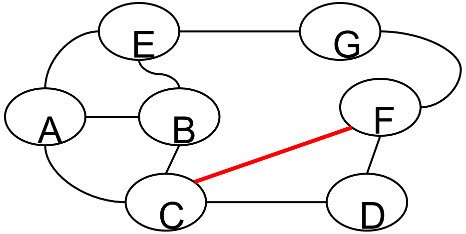


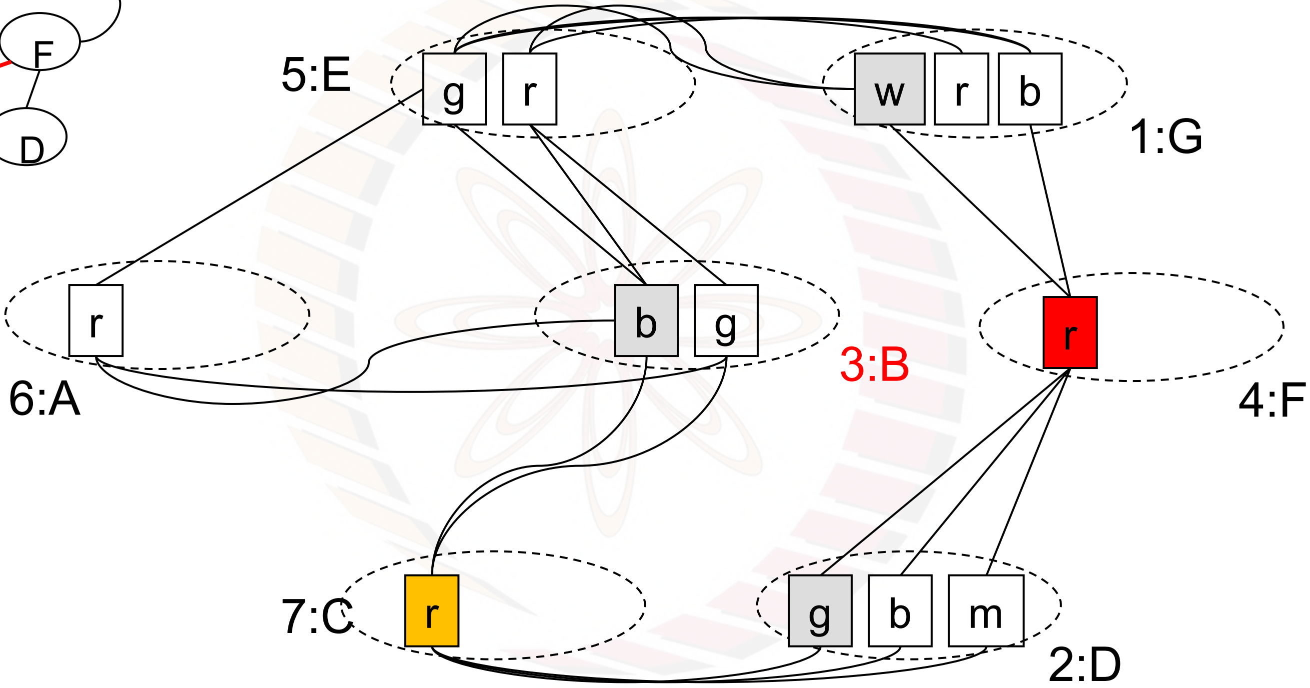
Forward Checking. G=w.

The value *w* is removed from nodes *E* and *F* related to *G*

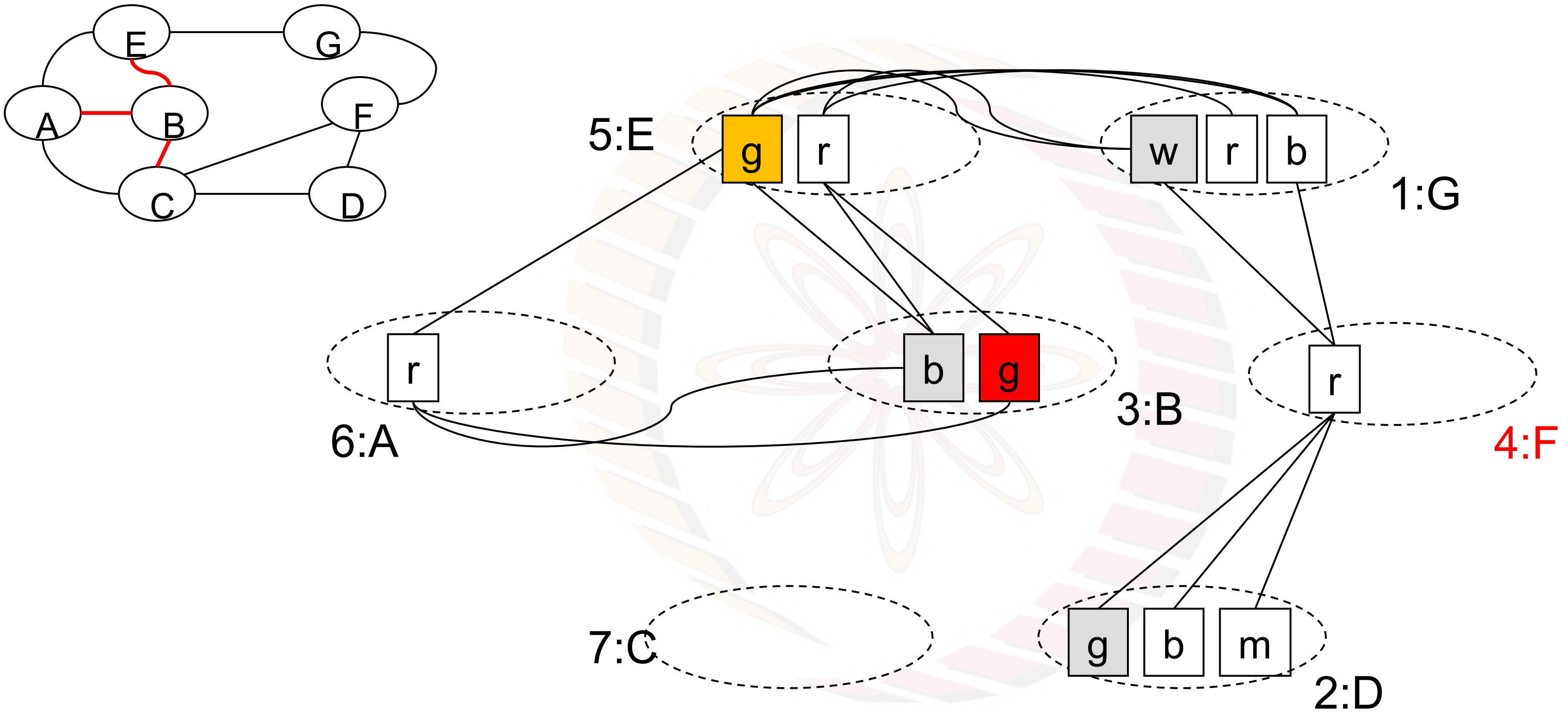


Forward Checking. G=w, D=g (no effect). Next B=b

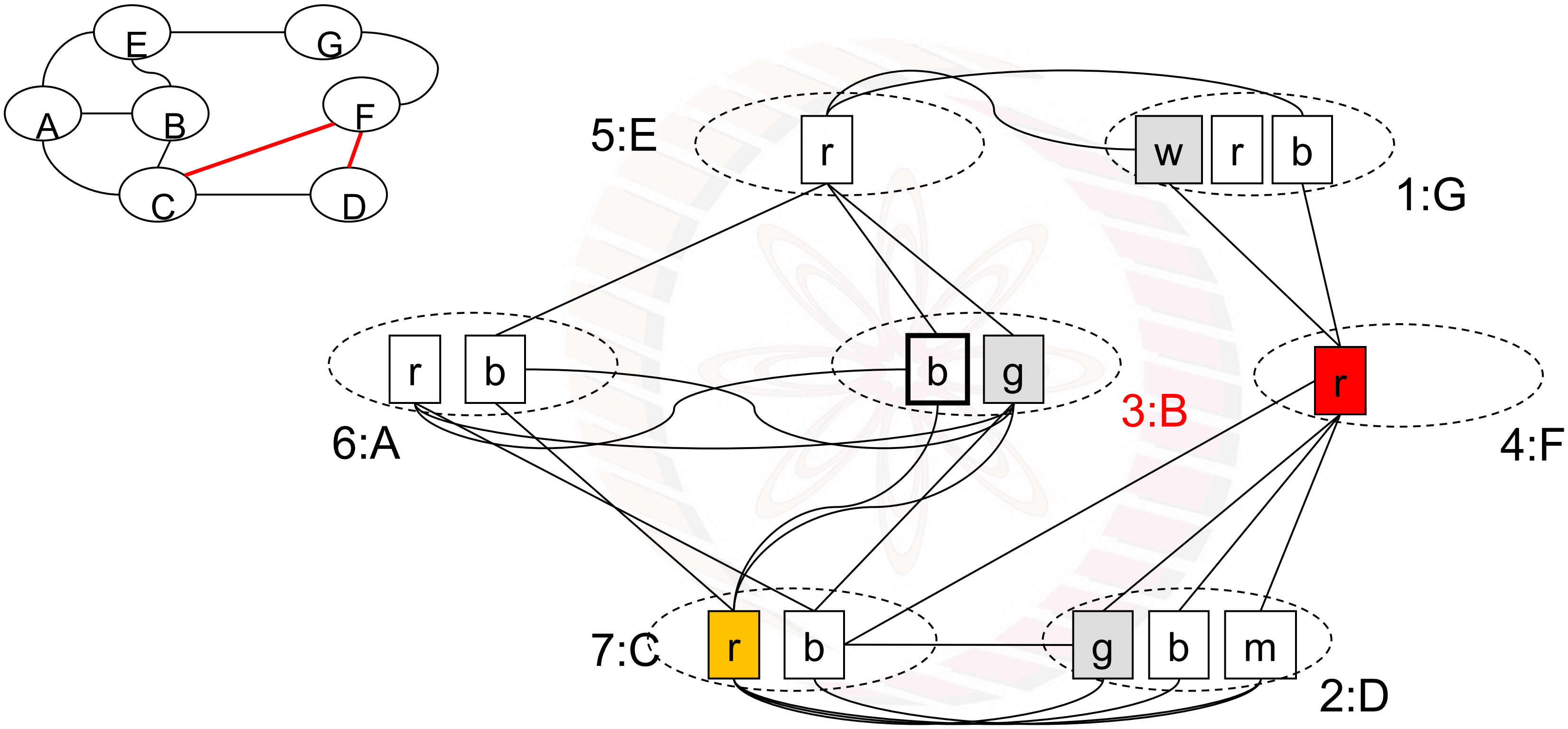




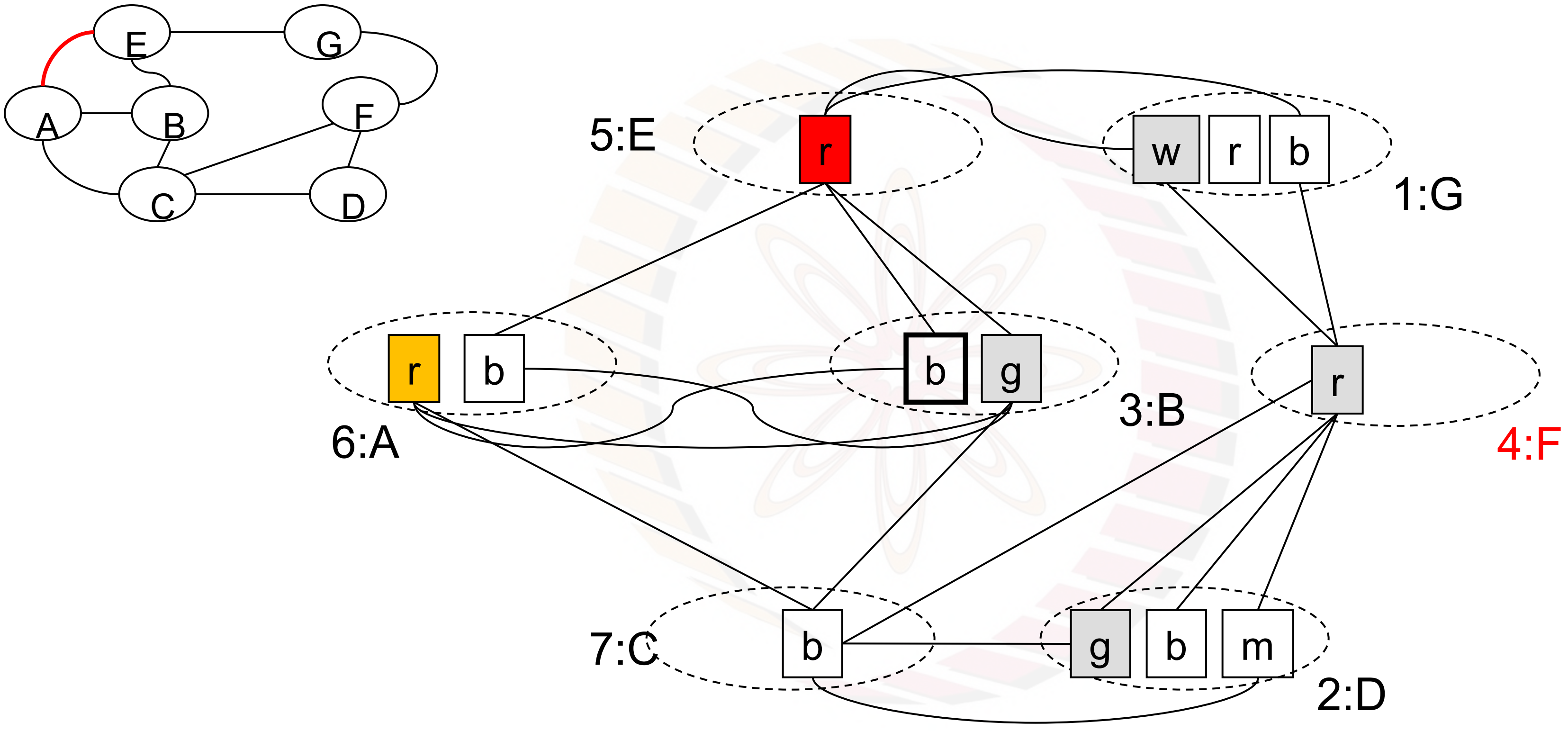
Forward Checking. G=w, D=g, B=b.
It *does not notice* that
edges AC and CF have become arc-inconsistent.
It carries on to F.



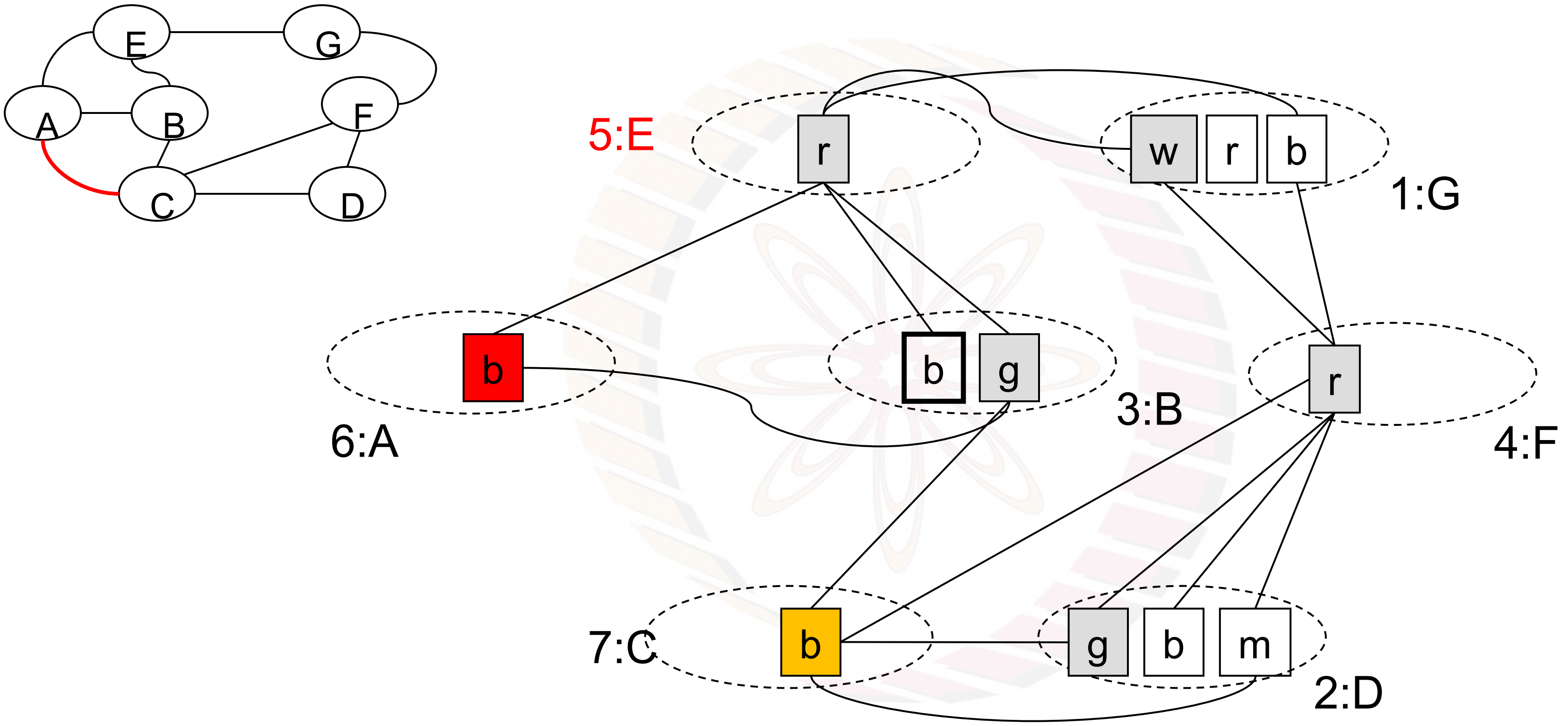
Forward checking *notices* that the domain of *C* has become empty and decides to backtrack. There is no other value for *F*, so it will try B=g now.



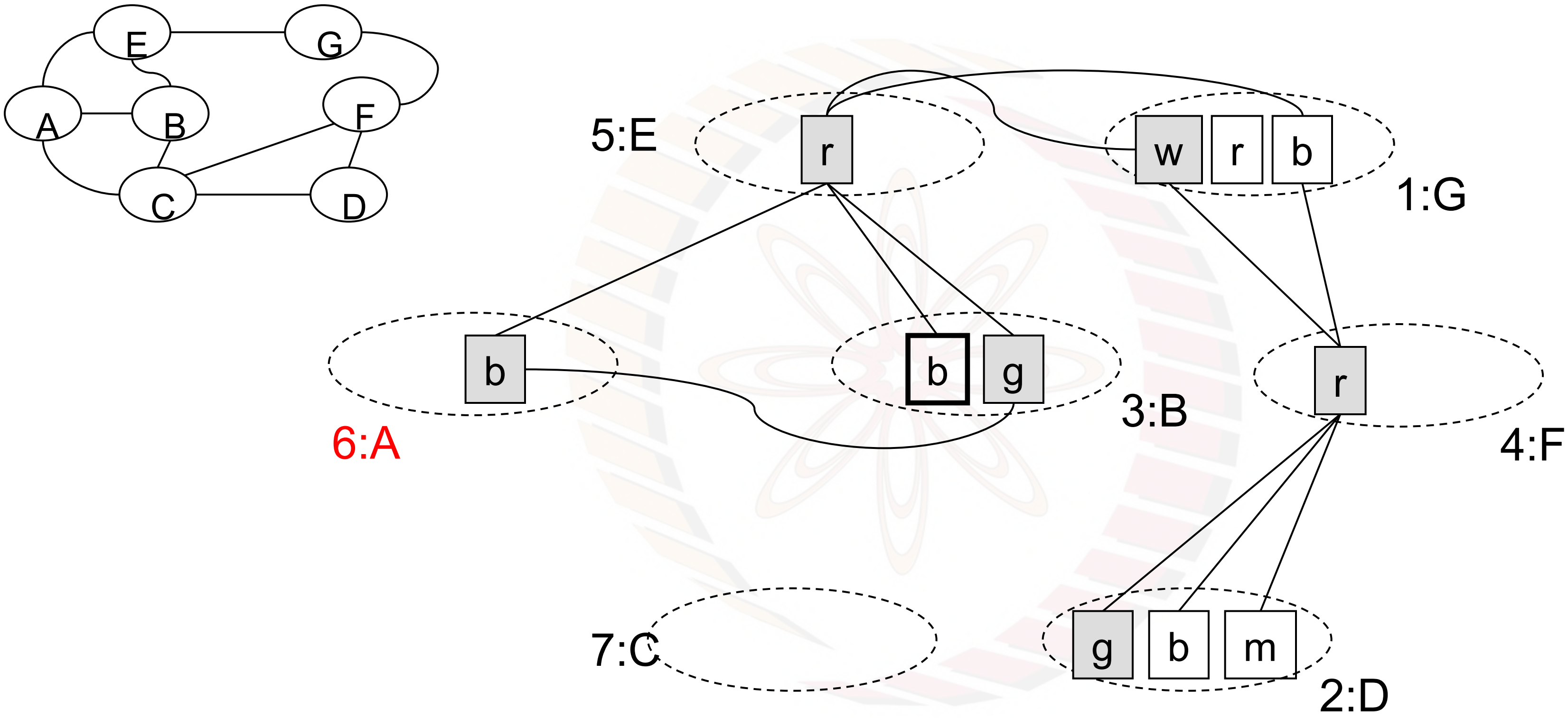
Forward Checking. G=w, D=g, after backtracking B=g.



Forward Checking. G=w, D=g, after backtracking B=g, F=r.



Again Forward Checking *does not notice* that AC is not arc consistent and *plods on* with A.



It tries A=b. Now it notices that domain of *C* has become empty and backtracks. Does not assign a value to *A*.

Increased propagation, reduced backtracking

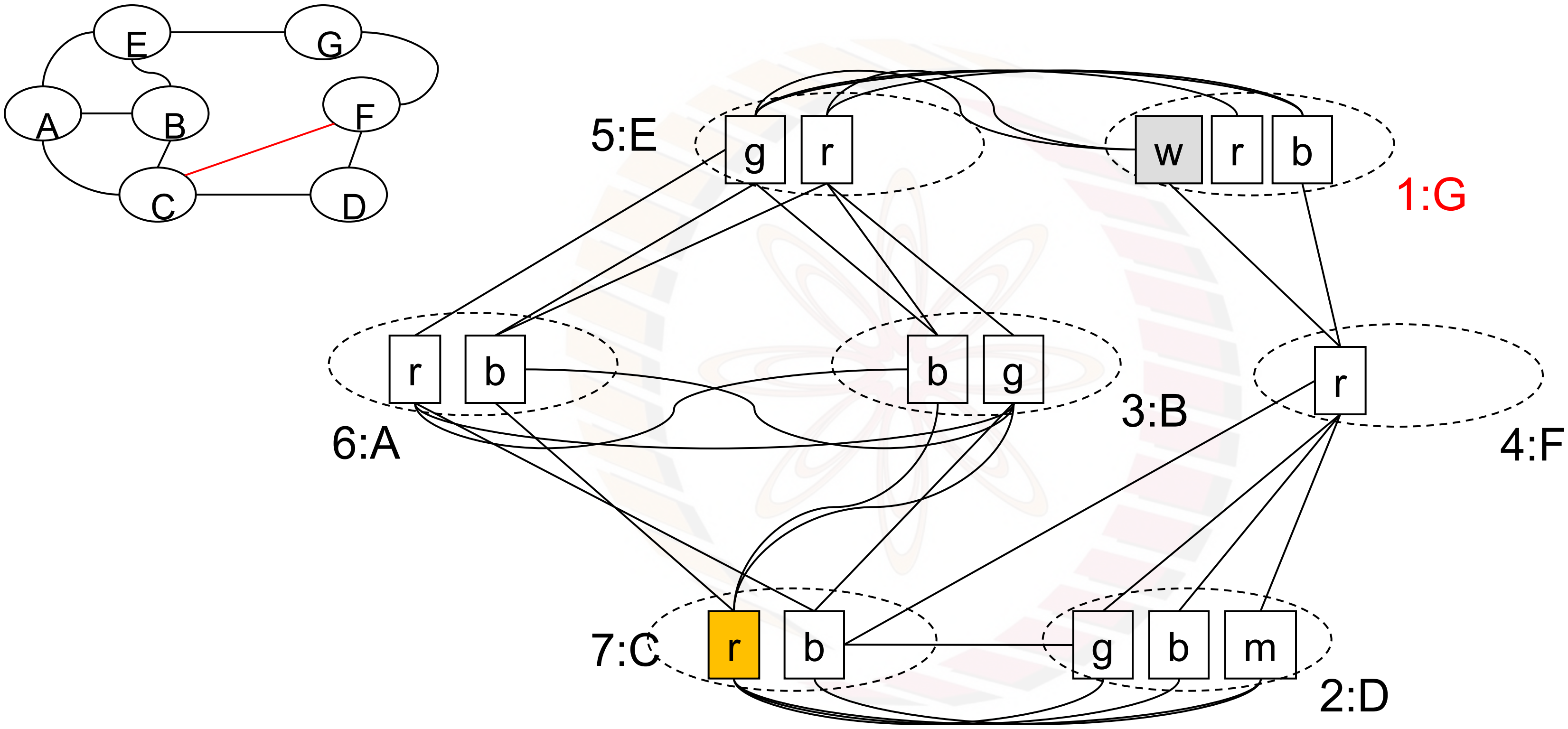
Forward Checking does the least amount of work in looking ahead

The following algorithms do increasingly more work

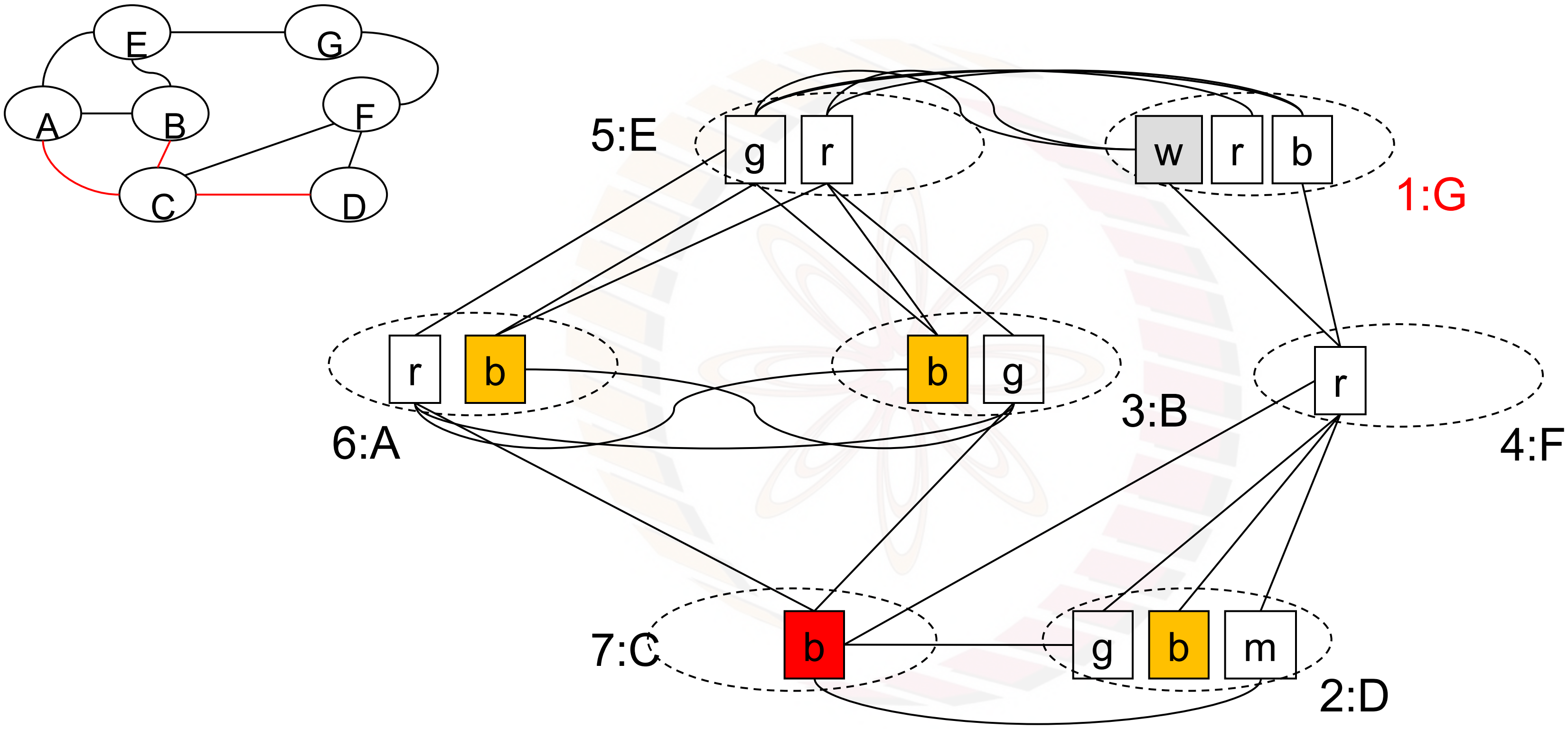
- Partial Lookahead
- Full Lookahead
- Arc Consistency Lookahead

The last one implements full arc consistency between the future variables

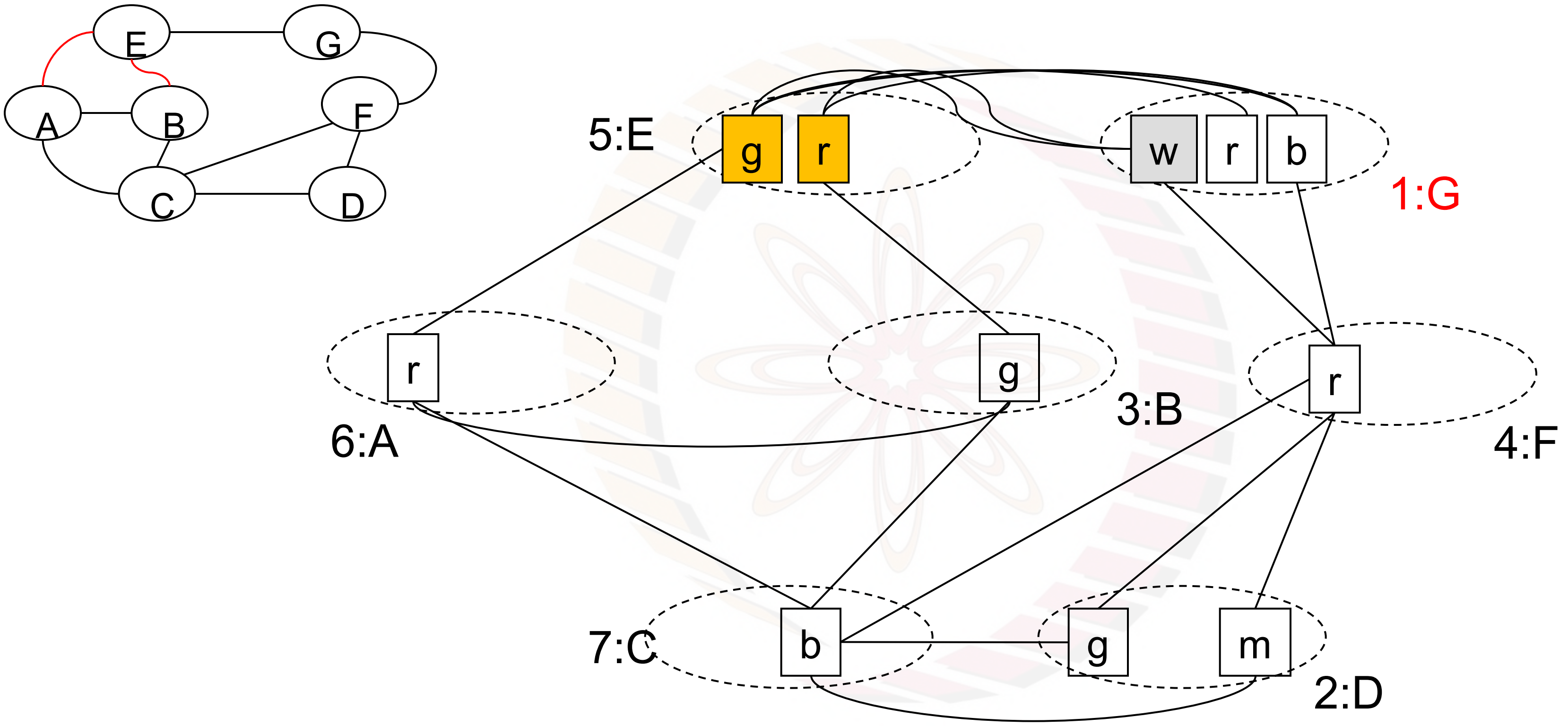
We take a small peek to wind up our discussion....



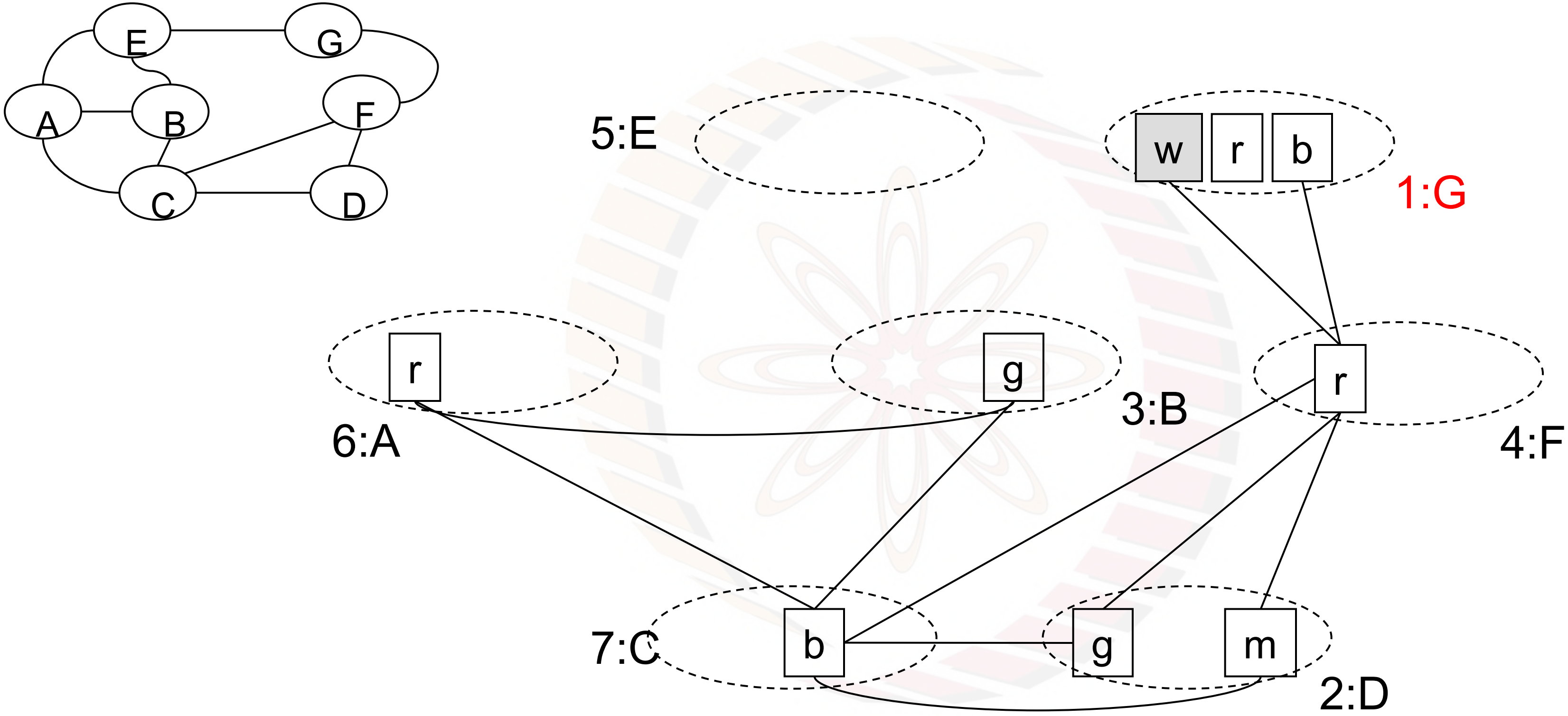
Full-AC. Does *full AC on the entire set of future variables*. G=w. Remove *w* from *F* and *E*. Remove *r* from *C* because not arc-consistent with *F*.



Full-AC. G=w. *Next* remove *b* from *A*, *B* and *D* (not arc consistent with *C*).



Full-AC. G=w. *Next* remove *r* from *E* (not consistent with *A*) and *g* from *E* (not consistent with *B*).



Full-AC. G=w. At this point *E* is empty. So G=w is not selected!

The Holy Grail...

- "The Holy Grail of Computer Science"
  - Eugene Freuder, University of Cork
  - one of the founding figures in Constraint Processing
  - in this paper* published in 1997
- *The user states the problem, and the computer solves it*
  - Artificial Intelligence!
- Constraint programming offers a unified framework in which search and reasoning can be combined
  - the more reasoning, the less search
  - look ahead methods integrate reasoning
  - look back methods integrate reasoning
  - look back methods exploit memorization

*Eugene C. Freuder, In Pursuit of the Holy Grail, *Constraints 2,* Springer, 57–61, 1997.

end

artificial intelligence
search methods for problem solving