Artificial & Computational Intelligence

DSE CLZG557

M3 : Game Playing & Constraint Satisfaction

Raja vadhana P

Assistant Professor,

BITS - CSIS

BITS Pilani

Pilani Campus

Course Plan

M1 Introduction to AI

M2 Problem Solving Agent using Search

M3 Game Playing, Constraint Satisfaction Problem

M4 Knowledge Representation using Logics

M5 Probabilistic Representation and Reasoning

M6 Reasoning over time, Reinforcement Learning

Module 3 : Part -1

Searching to play games

A. Minimax Algorithm

B. Alpha-Beta Pruning

C. Making imperfect real time decisions

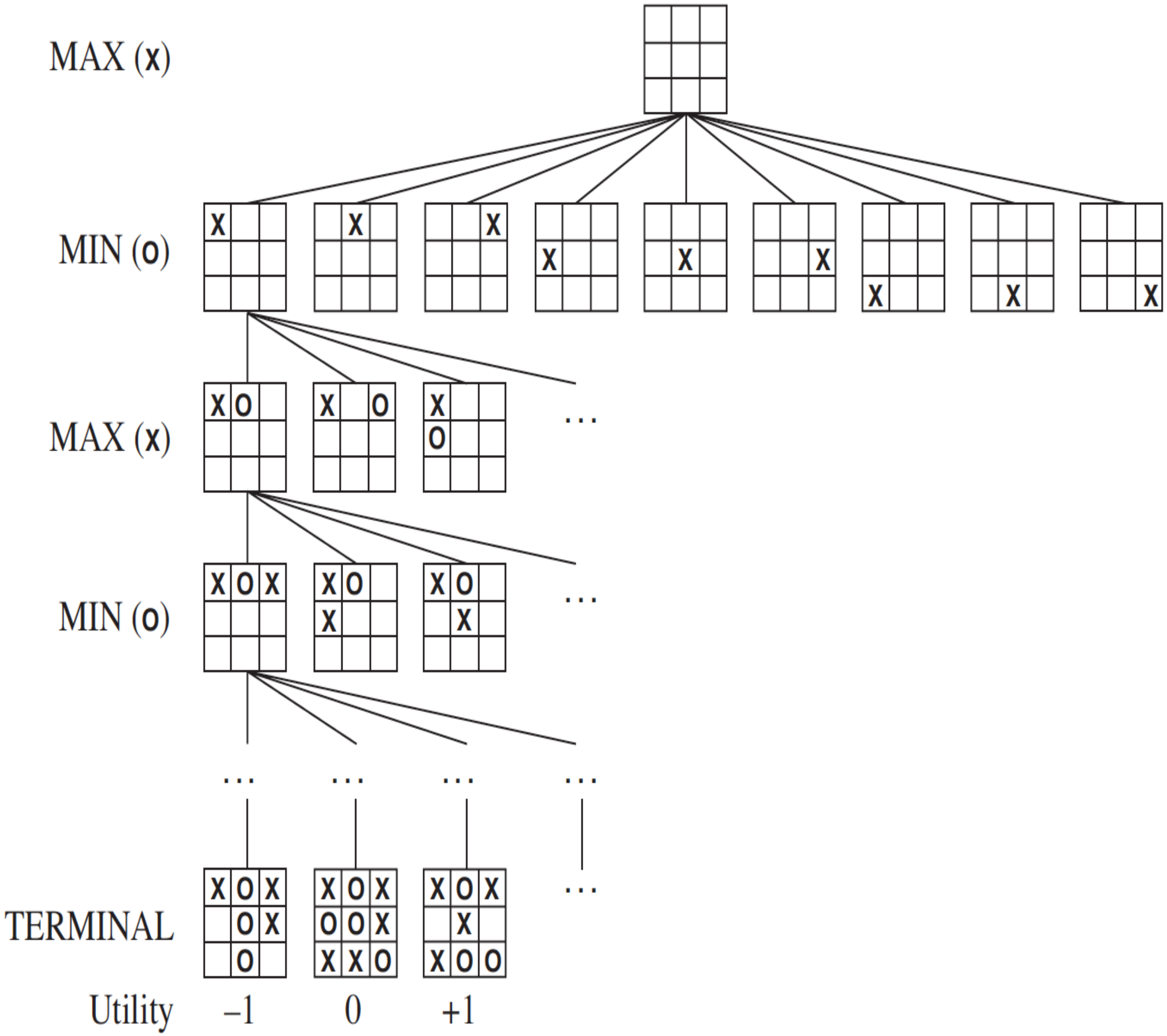
Problem Formulation

Games as Search Problem

PSA : Representation of Game:

INITIAL STATE: S0
PLAYER(s)
ACTIONS(s)
RESULT(s, a)
TERMINAL-TEST(s)
UTILITY(s, p)

Eg., Tic Tac Toe



<u>Assumption Task Environment:</u>

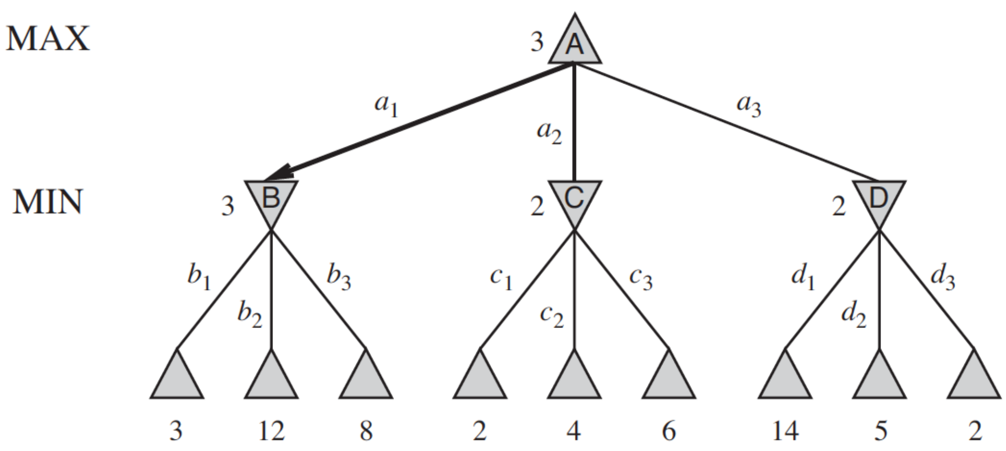
Static
Strategic
Multi agent
Fully observable
Sequential
Discrete

Making imperfect real time decisions

Computational Efficiency

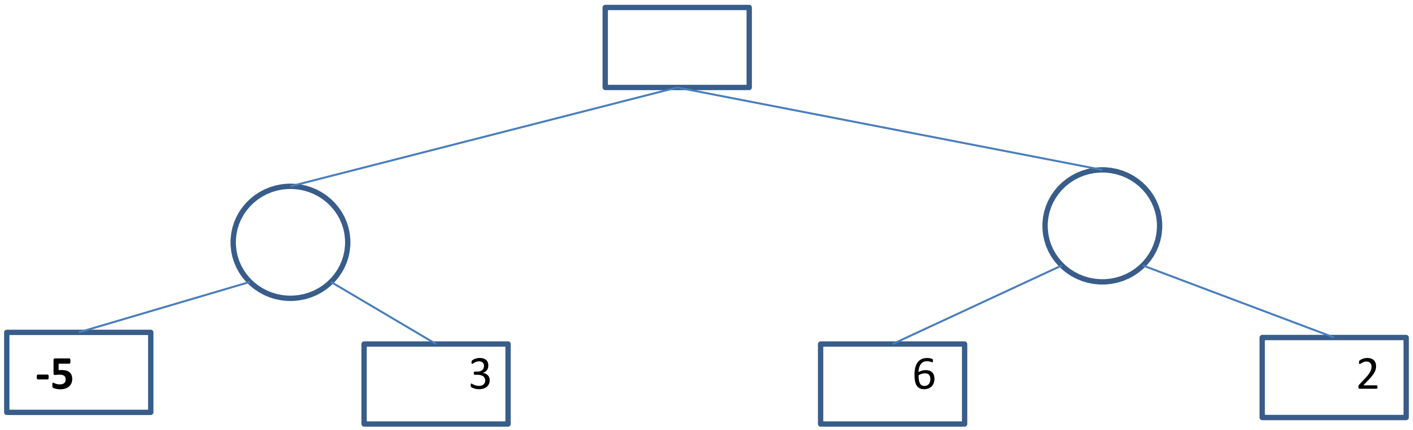
How to reduce the re-computation of the evaluators even with Alpha-Beta Pruning?

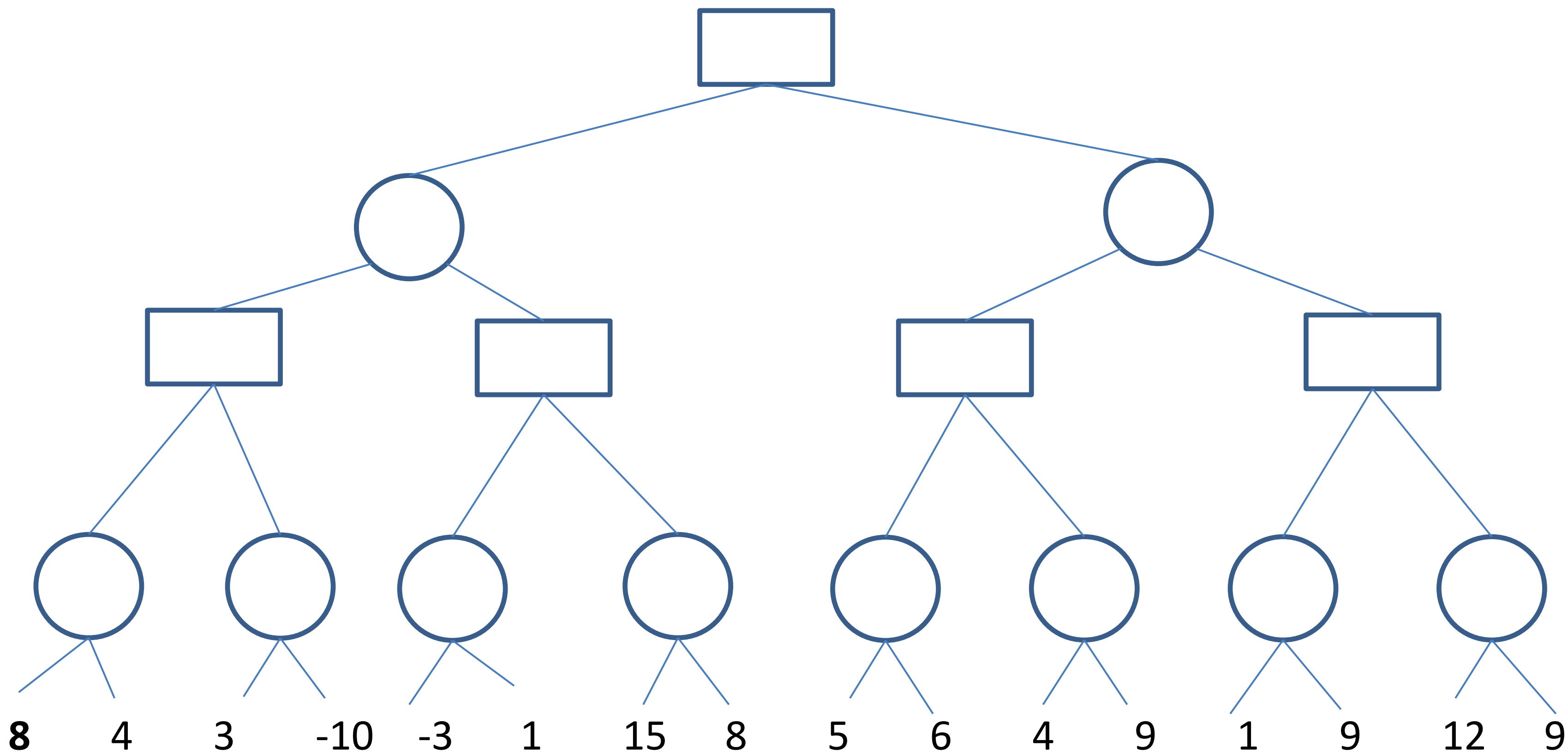
What if my capacity to look ahead is relatively shallow ?



Squares represent MAX nodes

Circles represent MIN nodes




8 4 3 -10 -3 1 15 8 5 6 4 9 1 9 12 9

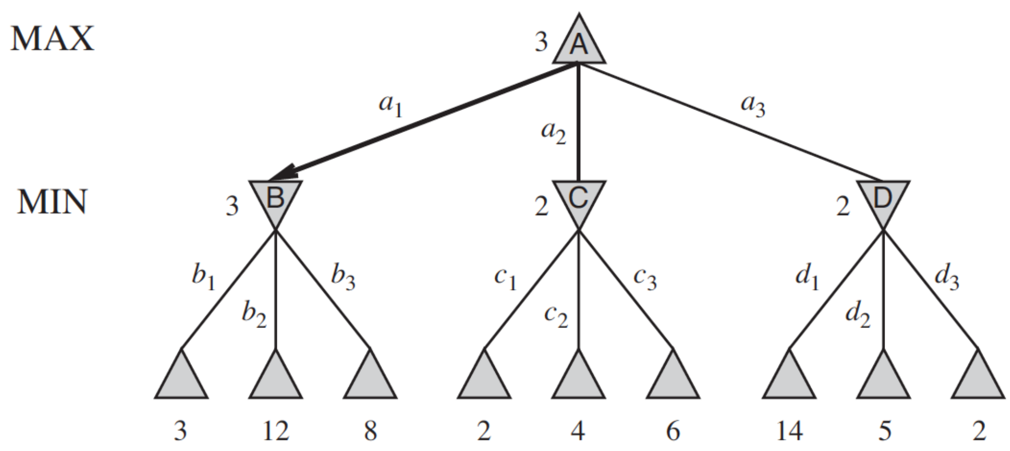
Computational Efficiency

How to reduce the re-computation of the evaluators even with Alpha-Beta Pruning?

What if my capacity to look ahead is relatively shallow ?

Idea:

Use heuristics to identify the state/node in which a move might bring drastic change in the value of the static evaluation function. Then keep the depth limit fix after the level thus delaying the generation of static value till that level.

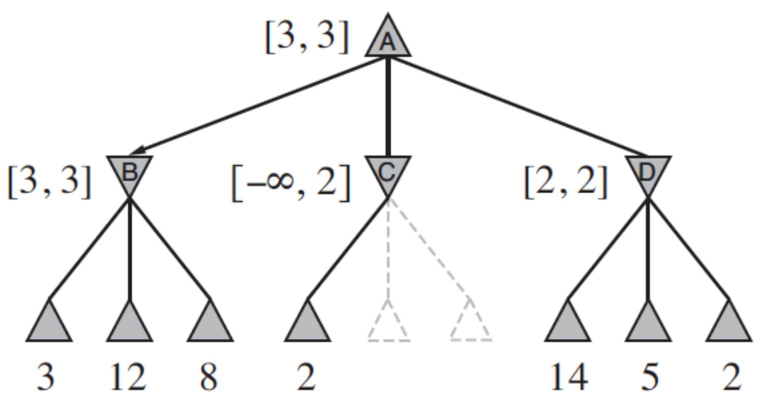


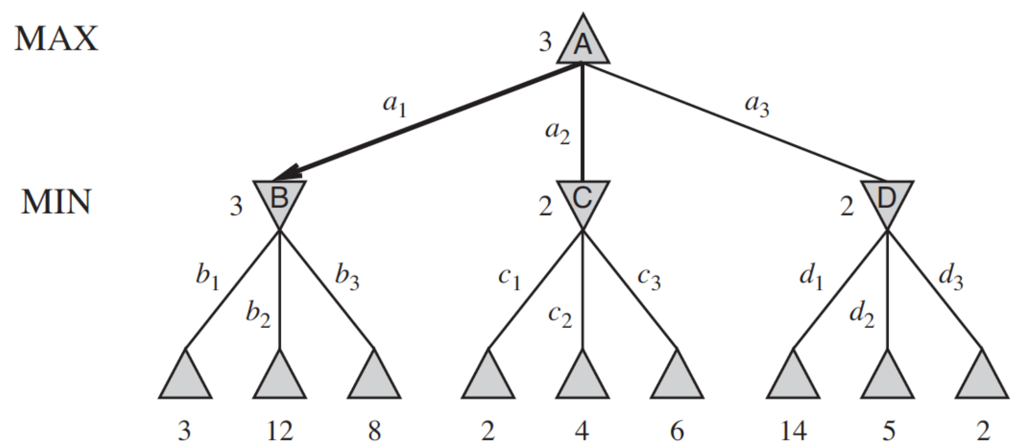
Computational Efficiency

How to reduce the re-computation of the evaluators even with Alpha-Beta Pruning?

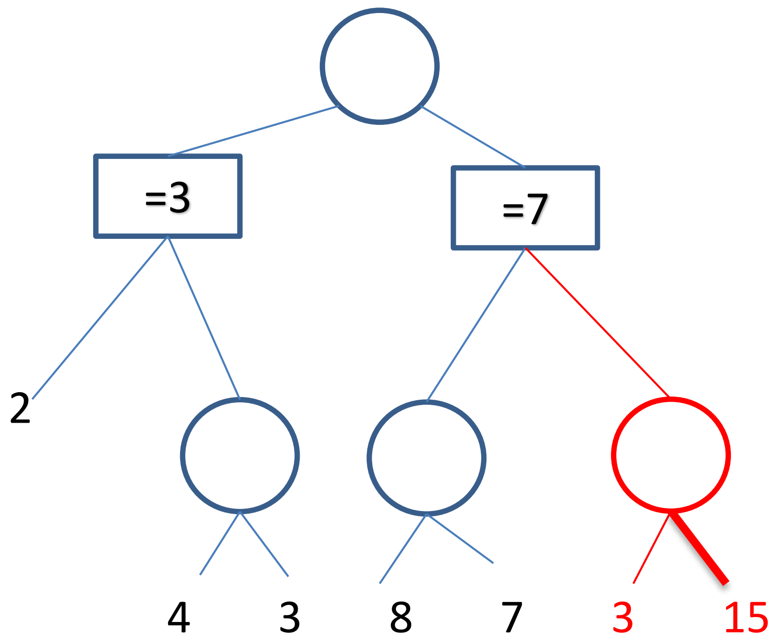
What if my capacity to look ahead is relatively shallow ?

How to reduce the move generations better along while doing Alpha-Beta Pruning?

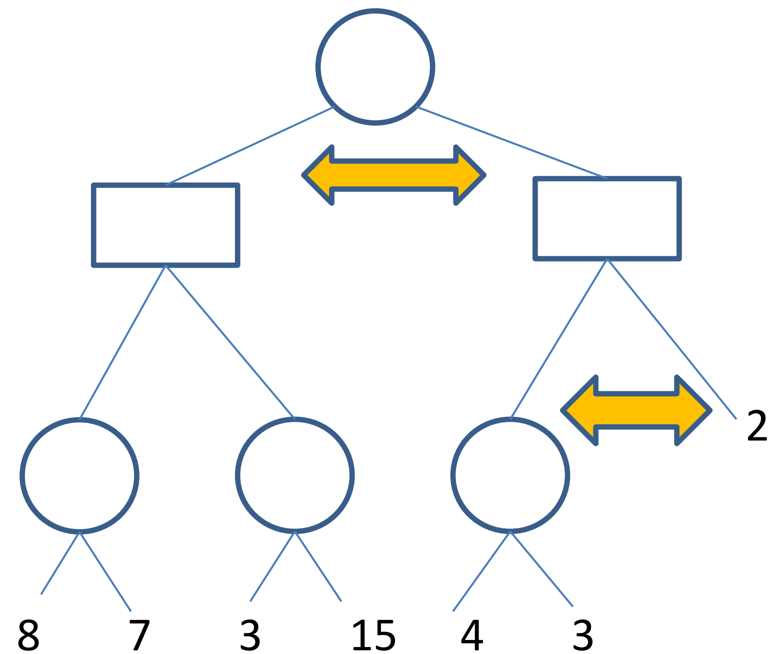




After Move Ordering



Before Move Ordering



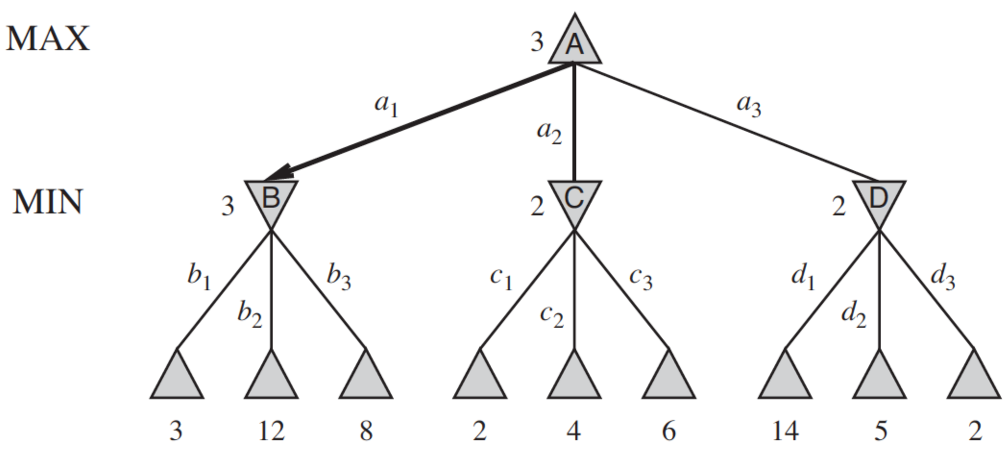
Computational Efficiency

How to reduce the re-computation of the evaluators even with Alpha-Beta Pruning?

What if my capacity to look ahead is relatively shallow ?

How to reduce the move generations better along while doing Alpha-Beta Pruning?

<u>Idea:</u>
Use heuristic of the game and prioritize game changing moves to be ordered as leftmost branch in the game tree.



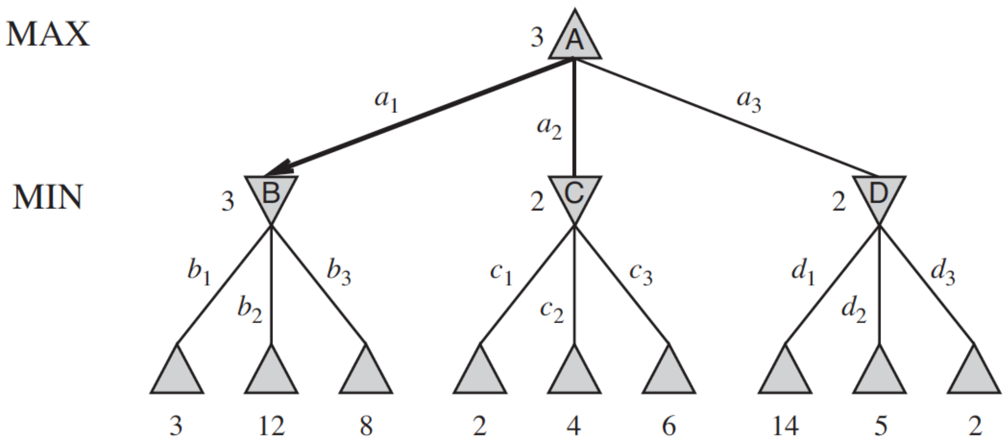
Computational Efficiency

How to reduce the re-computation of the evaluators even with Alpha-Beta Pruning?

What if my capacity to look ahead is relatively shallow ?

How to reduce the move generations better along while doing Alpha-Beta Pruning?

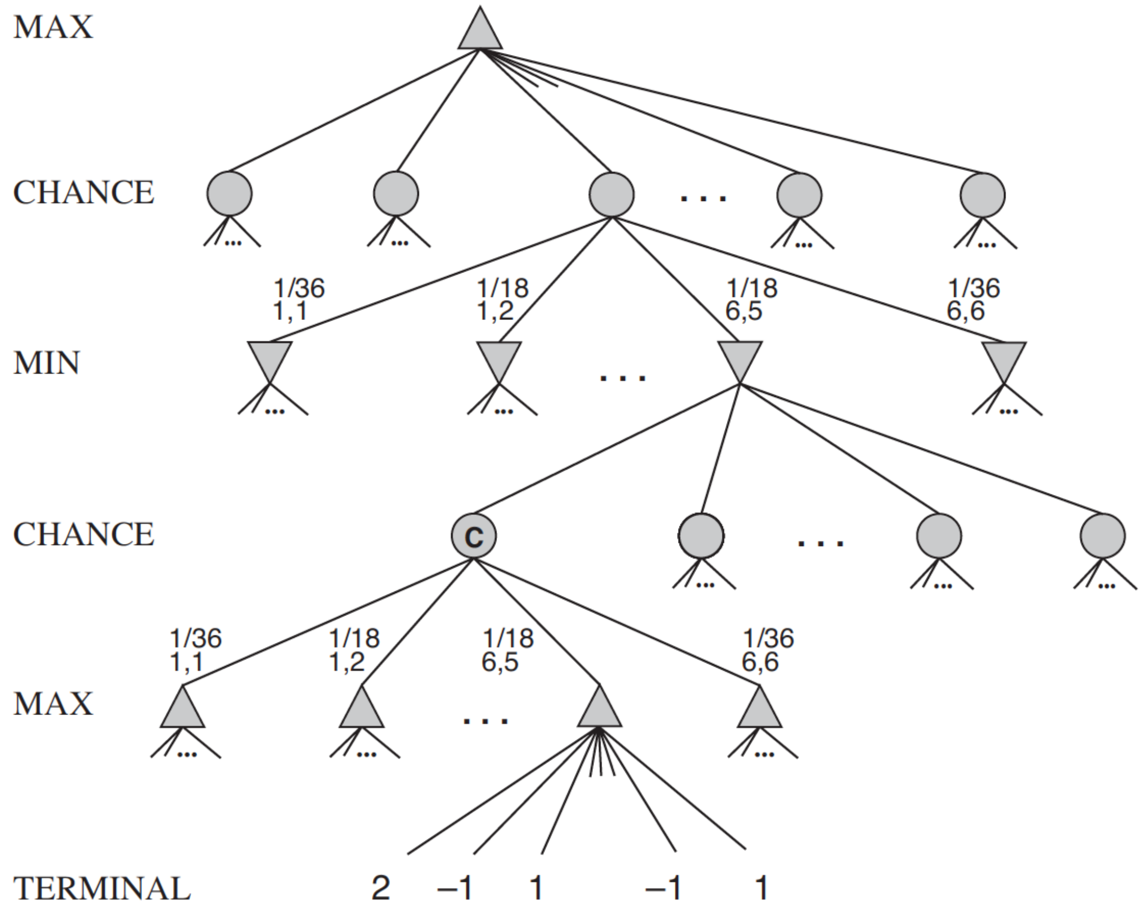
How games can be designed to handle imperfect decisions in real-time?

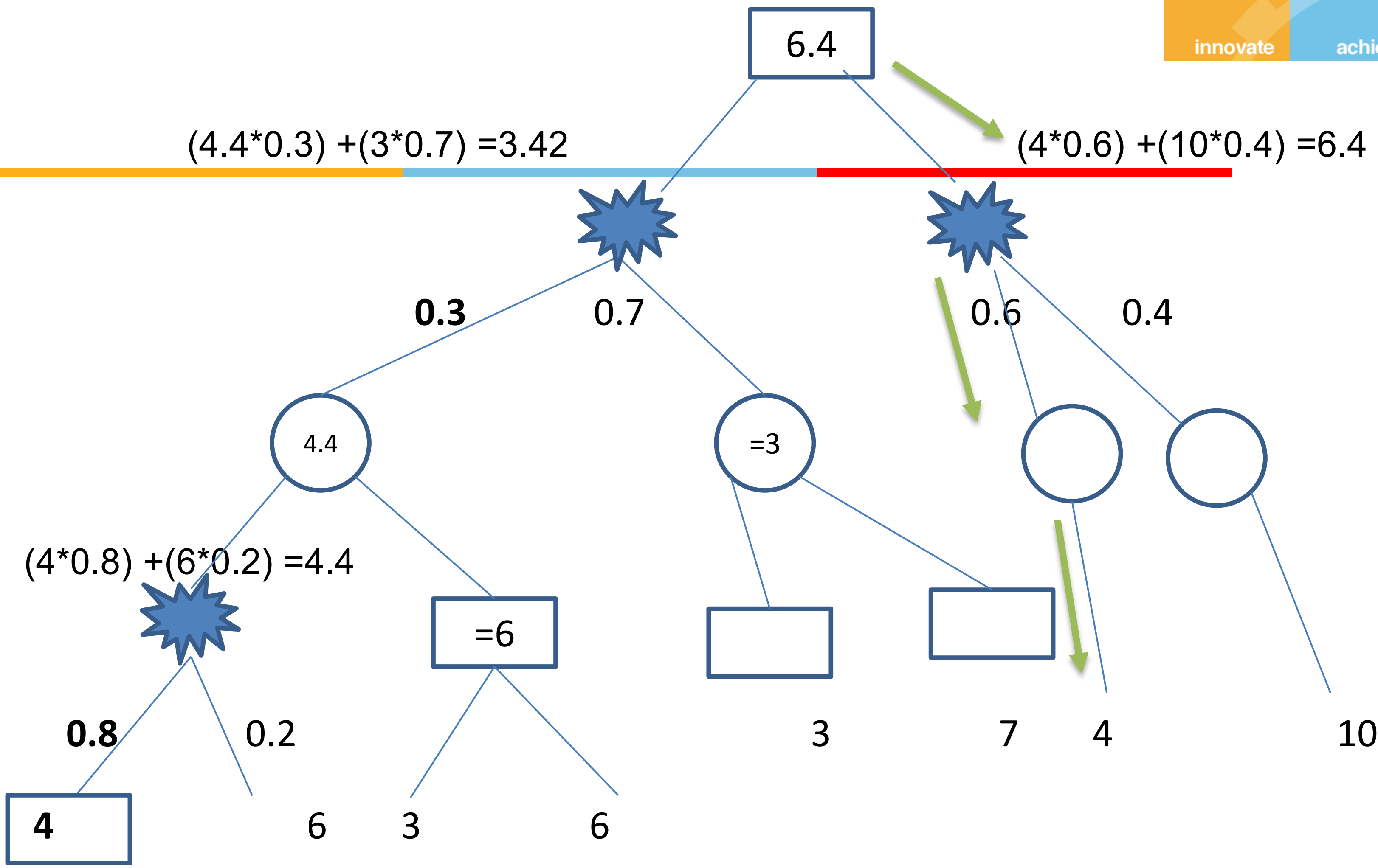


Computational Efficiency

Idea : Chance Node:

Holds the expected values that are computed as a sum of all outcomes weighted by their probability (of dice roll)



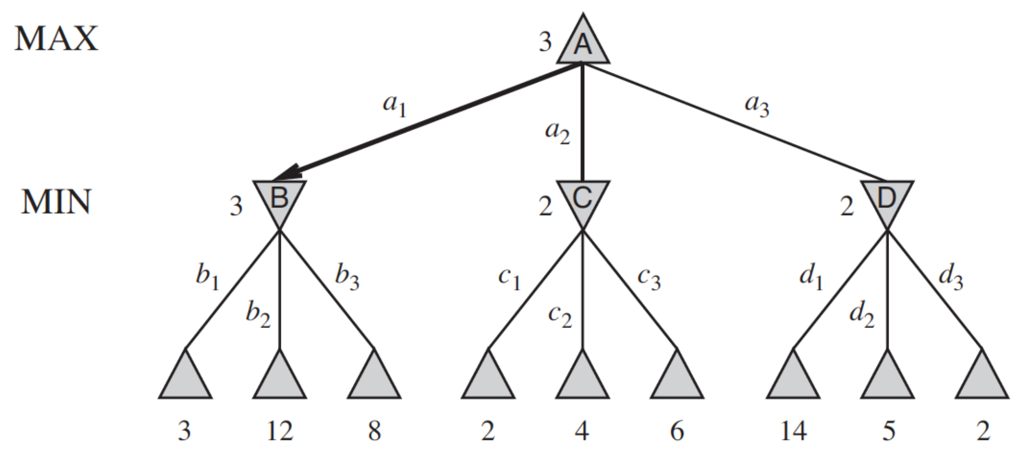


6.4
(4.4*0.3) +(3*0.7) =3.42
(4*0.6) +(10*0.4) =6.4
0.3
0.7
0.6
0.4
4.4
=3
(4*0.8) +(6*0.2) =4.4
=6
0.8
0.2
3
7
4
10
4
6
3
6

Computational Efficiency

<u>Credit Assignment Problem:</u>

The credit assignment problem concerns determining how the success of a system's overall performance is due to the various contributions of the system's components



Computational Efficiency

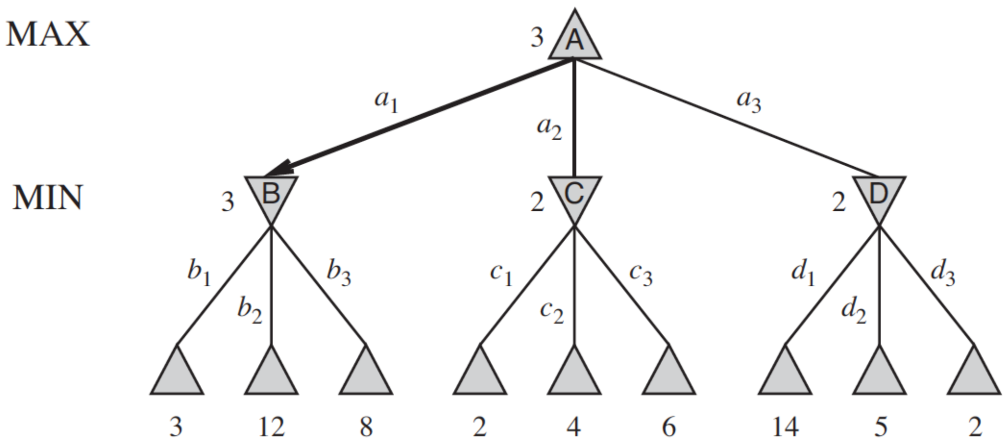
Is it possible to reduce the game tree size further in cases where the number of possible moves are large but still finite & game is finish able?

Monte Carlo Tree Search:

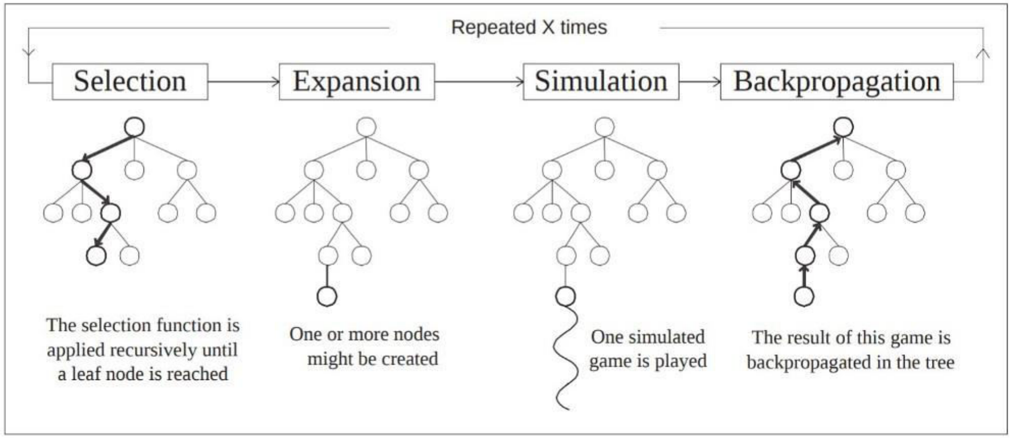
For each possible legal moves , simulate k random games .

Select the move that has most number of wins

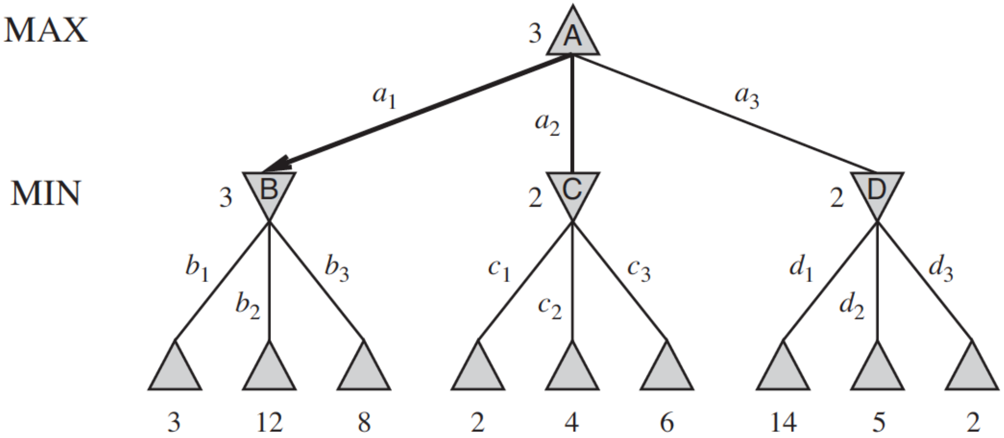
<u>Idea:</u>



Computational Efficiency



Source Credit : MCTS algorithm, diagram from Chaslot (2006)



Module 3 : Part -2

Constraint Satisfaction Problem

A. Formulating a CSP problem

B. Constraint propagation

C. Local search for CSP

Constraint Satisfaction Problem

A problem is solved when each variable has a value that satisfies all the constraints on that variable

Problem Definition:

A Constraint Satisfaction Problem consists of three components X, D and C

- X – set of all variables $\{X_1, X_2, X_3, \dots, X_n\}$
- D – set of domains, one for each variable $\{D_1, D_2, D3, \dots, D_n\}$
- Non empty domain of possible values for each variable
- C – set of constraints that specify allowable combinations of values

Problem Formulation

- Each domain would have a set of values that are allowed
  - D_1 would have {1, 2, 4, 5, 7} meaning X_1 can take only those values

- Each Constraint is a pair of <scope, relation>
  - Where scope = tuple of variables
  - Relation = relation defining the possible values of that variables
  - E.g., if variable X_1 and X_2 cannot take same values
  - $<(X_1, X_2), X_1 \neq X_2>$
  - E.g., If (SA = blue), then its five neighbors cannot have ”blue”
    - Reducing the possible search space to $2^5 = 32$ instead of the original $3^5 = 243$

Problem Formulation

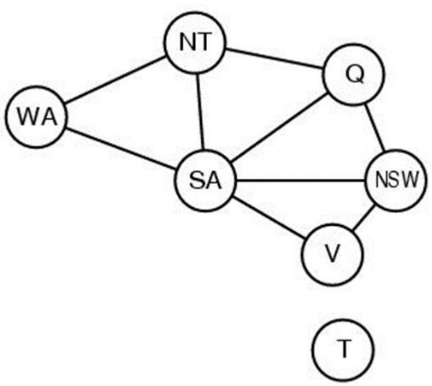
- A state is defined as an assignment of values to some or all variables.
- Assignment
  - Consistent assignment: assignment does not violate the constraints
  - An assignment is complete when every variable is assigned a value.
  - A solution to a CSP is a complete assignment that satisfies all constraints.
- Some CSPs require a solution that maximizes an objective function.

Applications:

- Scheduling problems
- Job shop scheduling
- Map colouring

Problem Formulation

Map Coloring Problem



Variables

WA, NT, Q, NSW, V, SA, T

Domain

D_i = {red , green, blue}
D_i = {R, G, B}

Constraints

WA ≠ NT

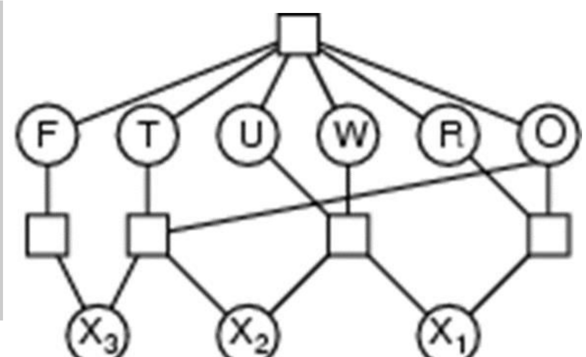
(WA,NT) =
{(R,G), (R,B), (G,R), …}

Solution

{WA = R, NT = G,Q = R,NSW = G, V = R, SA =B ,T = G }

Crypt arithmetic Problem

```
  T W O
+ T W O
-------
F O U R
```



Crypt Arithmetic Problem – Example

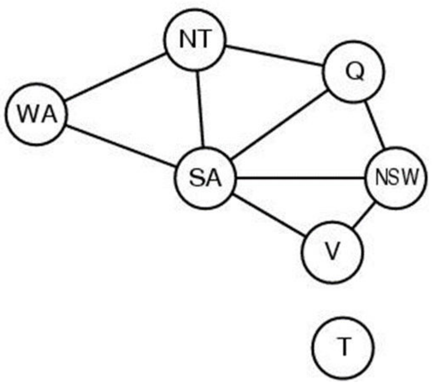
1	1	0	1	
		E	A	T
		8	1	9
	+ T	H	A	T
	9	2	1	9
= A	P	P	L	E
1	0	0	3	8

EAT THAT APPLE & Get the PLATE

PLATE : 03198

Problem Formulation

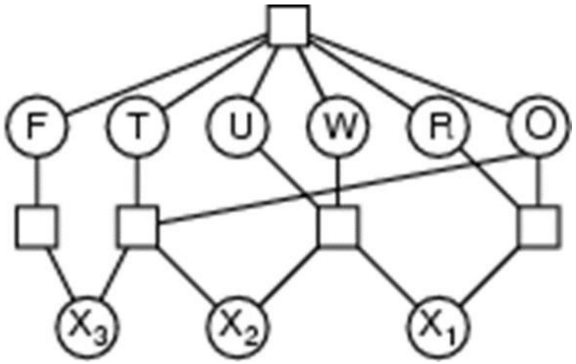
Map Coloring Problem



Variables
F T U W R O, X1 X2 X3
Domain
{0,1,2,3,4,5,6,7,8,9} {0,1}
Constraints
F≠ O≠ R ≠T ≠U≠W O + O = R + 10 · X1 X1 + W + W = U + 10 · X2 X2 + T + T = O + 10 · X3 X3 = F, T ≠ 0, F ≠ 0
Solution

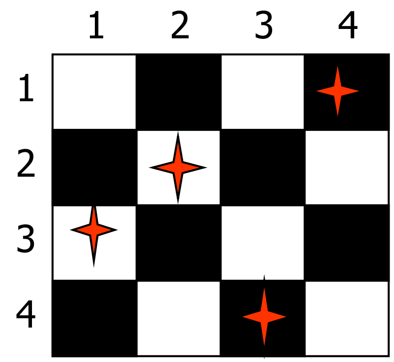
Crypt arithmetic Problem

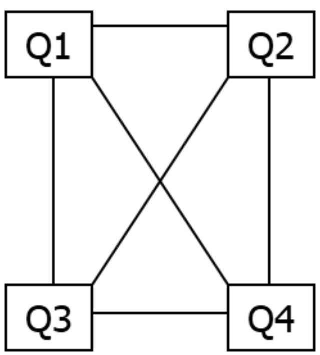
```
   T W O
 + T W O
 -------
 F O U R
```



Problem Formulation

N-Queen





Variables
[Xij]

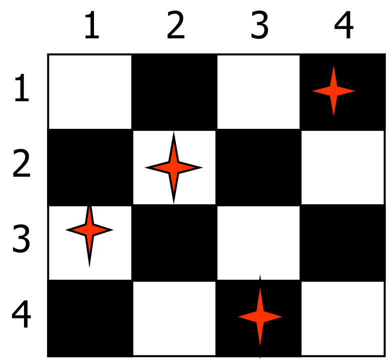
Domain
{0,1}

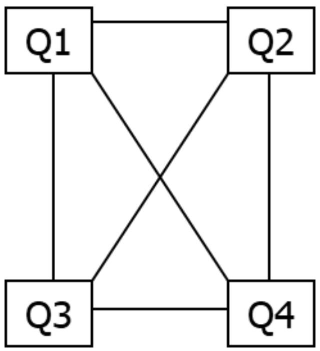
Constraints
(Xij, Xik) = {(0,0), (0,1), (1,0)} (Xij, Xkj) = {(0,0), (0,1), (1,0)} (Xij, Xi+k, j+k) = {(0,0), (0,1), (1,0)} (Xij, Xi+k,j-k) = {(0,0), (0,1), (1,0)} $\sum_{ij} Xij = N$

Solution
$X_{3,1}=1$, $X_{2,2}=1$, $X_{4,3}=1$, $X_{1,4}=1$

Problem Formulation

N-Queen

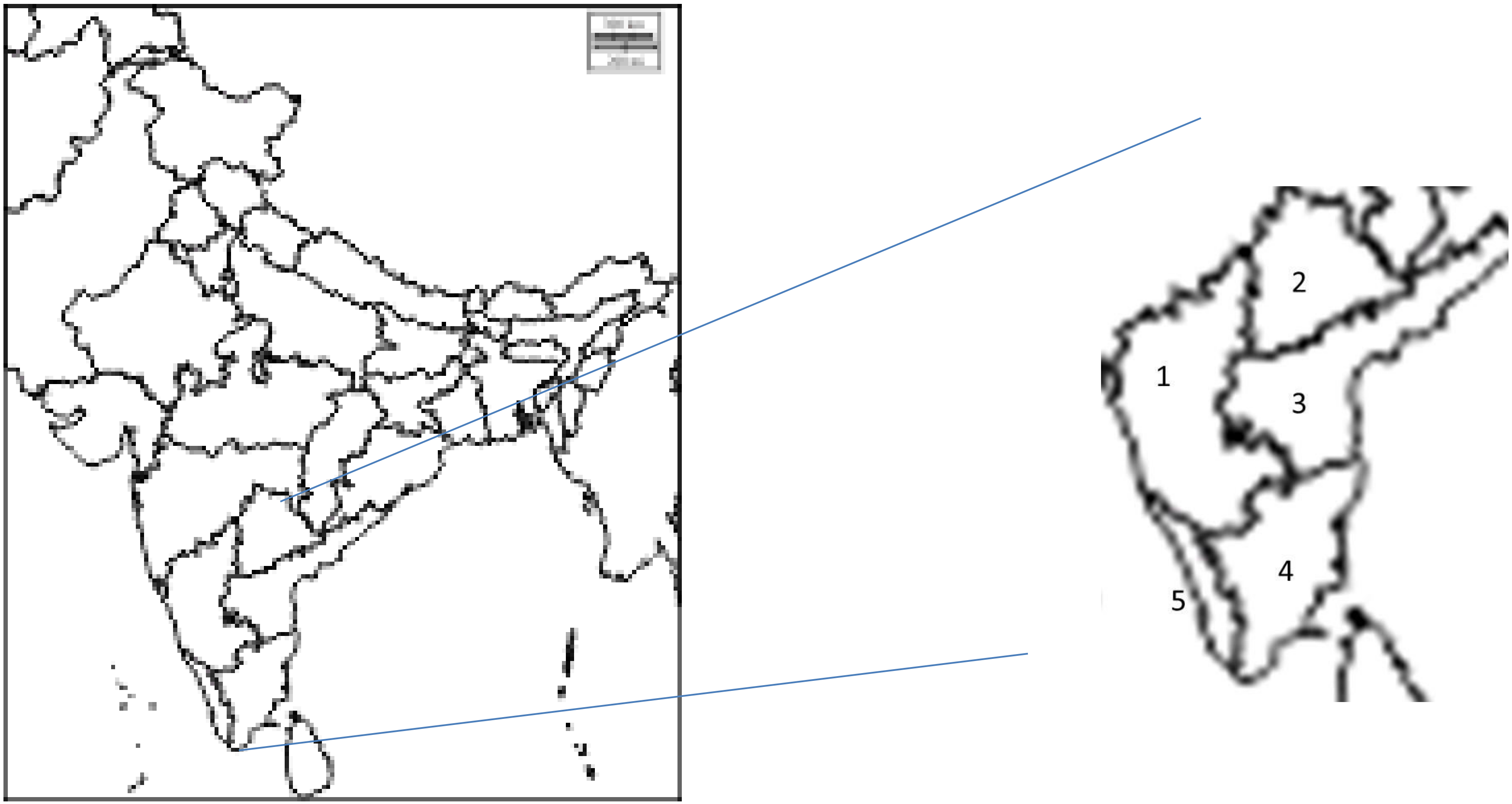




Variables
Q1, Q2, Q3, Q4.. QN
Domain
{1,2,.....N}
Constraints
(Q1,Q2)={(1,3), (1,4), (2,4)...}
Solution
Q1=3, Q2=2, Q3=4, Q4=1

Introduction to Constraint Propagation

Constraint Satisfaction Problem



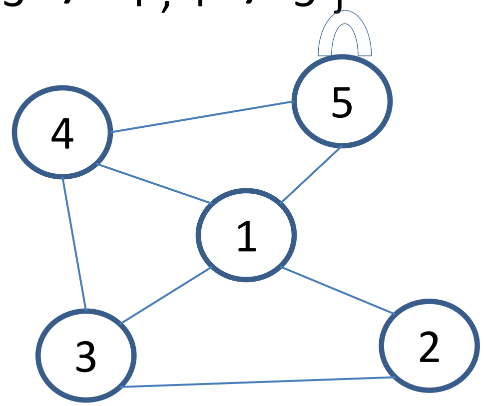
Constraint Satisfaction Problem

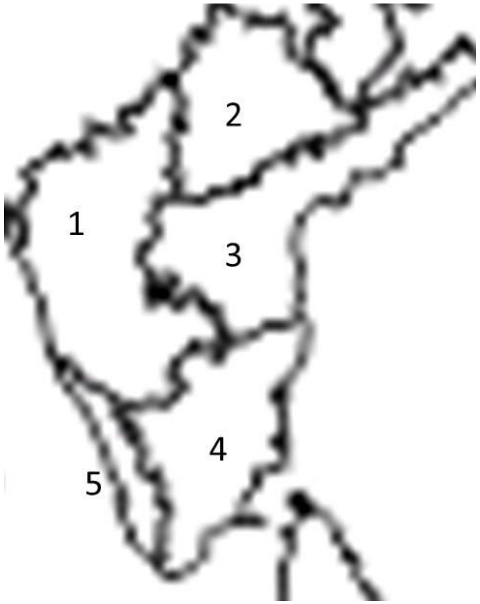
Variables = {1, 2, 3, 4, 5}
Domain = { R , G, B, Y }
Constraints =

{ 5 ≠ R ,

1` ≠ 2`, 1` ≠ 3`, 1` ≠ 4` , 1` ≠ 5`,

2` ≠ 3` , 3` ≠ 4`, 4` ≠ 5`}





Objective: Color the marked states with available colors (from the domain set) such that no neighboring states share the same color.
Another restriction(constraint) is that state coded as 5 should not have Red(R) color

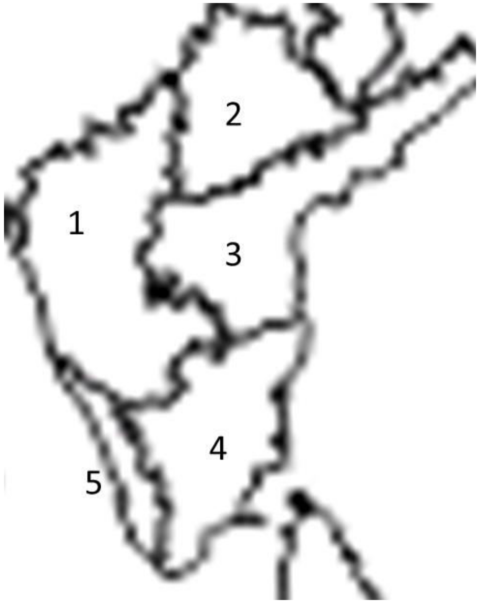
Constraint Satisfaction Problem

Variables = {1, 2, 3, 4, 5}
Domain = { R , G, B, Y }
Constraints =

{ 5 ≠ R ,

1` ≠ 2`, 1` ≠ 3`, 1` ≠ 4` , 1` ≠ 5`,

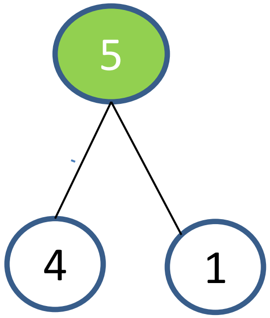
2` ≠ 3` , 3` ≠ 4`, 4` ≠ 5`}

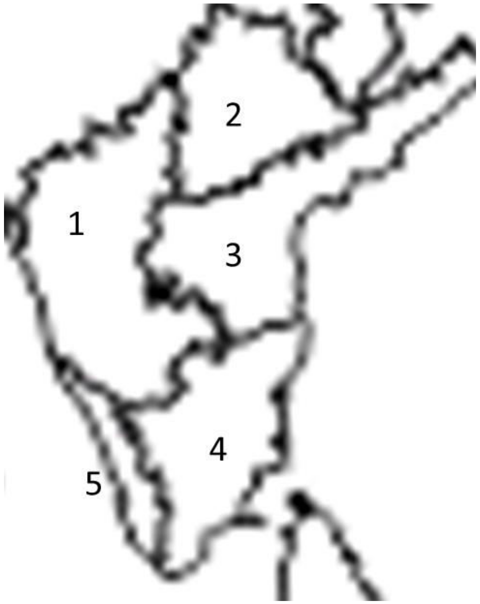


Problem Formulation
<u>Initial State</u> : Empty Assignment .
<u>Successor Function</u> : Consistent current assignment
<u>Goal Test</u> : Complete Consistent Assignment as of this state
<u>Path Cost</u> : Every backtracking = 10 penalty or Every Dead End = 5

Constraint Satisfaction Problem

1	2	3	4	5
				G





Constraints =

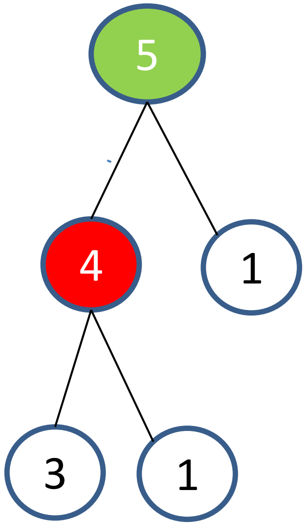
{ 5 ≠ R ,

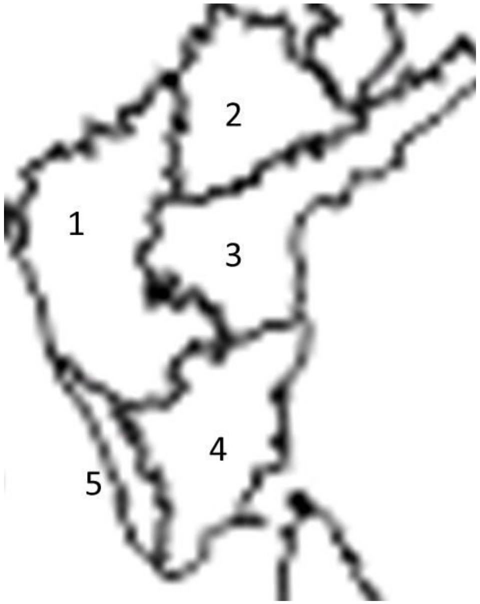
1` ≠ 2`, 1` ≠ 3`, 1` ≠ 4` , 1` ≠ 5`,

2` ≠ 3` , 3` ≠ 4`, 4` ≠ 5`}

Constraint Satisfaction Problem

1	2	3	4	5
				G
			R	G





Constraints =

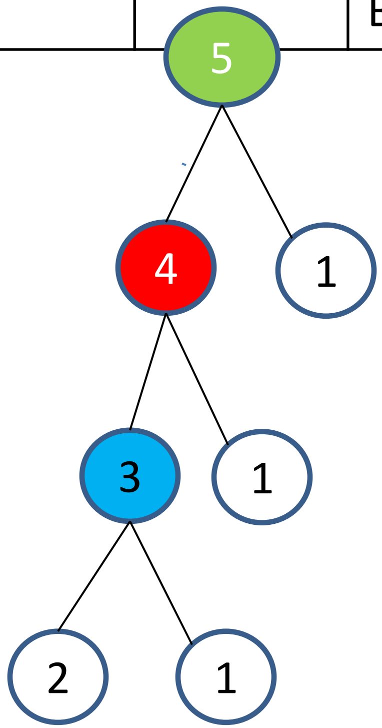
{ 5 ≠ R ,

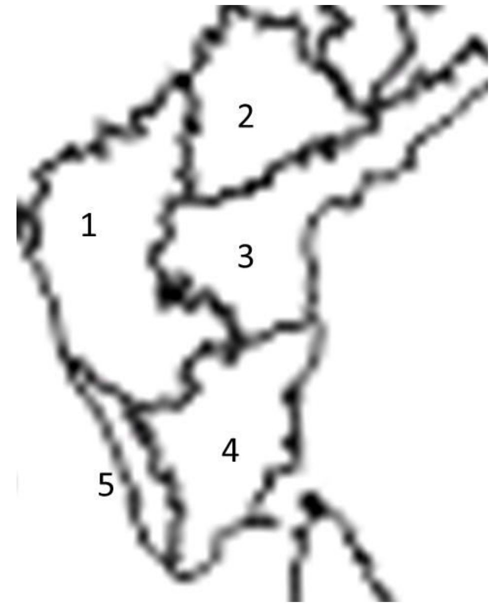
1` ≠ 2`, 1` ≠ 3`, 1` ≠ 4` , 1` ≠ 5`,

2` ≠ 3` , 3` ≠ 4`, 4` ≠ 5`}

Constraint Satisfaction Problem

1	2	3	4	5
				G
			R	G
		B	R	G





Constraints =

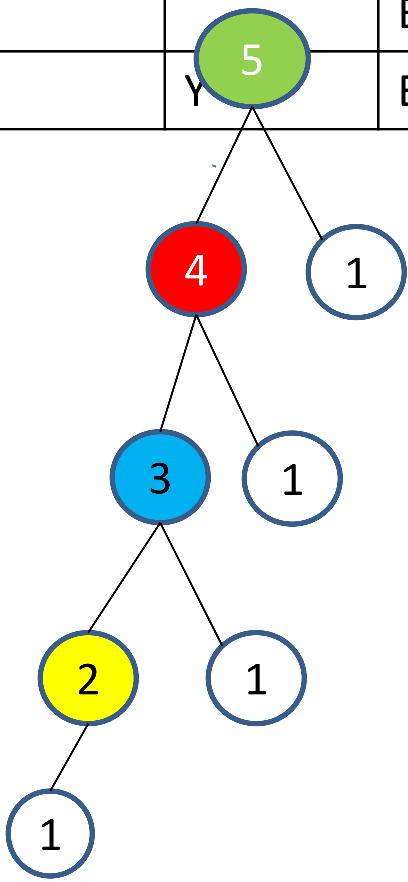
{ 5 ≠ R ,

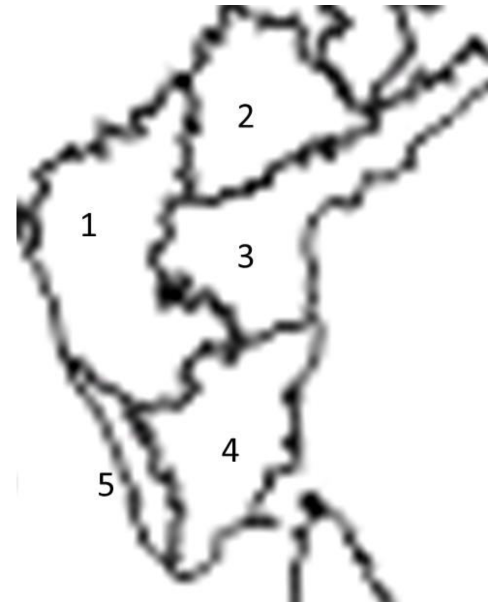
1` ≠ 2`, 1` ≠ 3`, 1` ≠ 4` , 1` ≠ 5`,

2` ≠ 3` , 3` ≠ 4`, 4` ≠ 5`}

Constraint Satisfaction Problem

1	2	3	4	5
				G
			R	G
		B	R	G
	Y	B	R	G





Constraints =

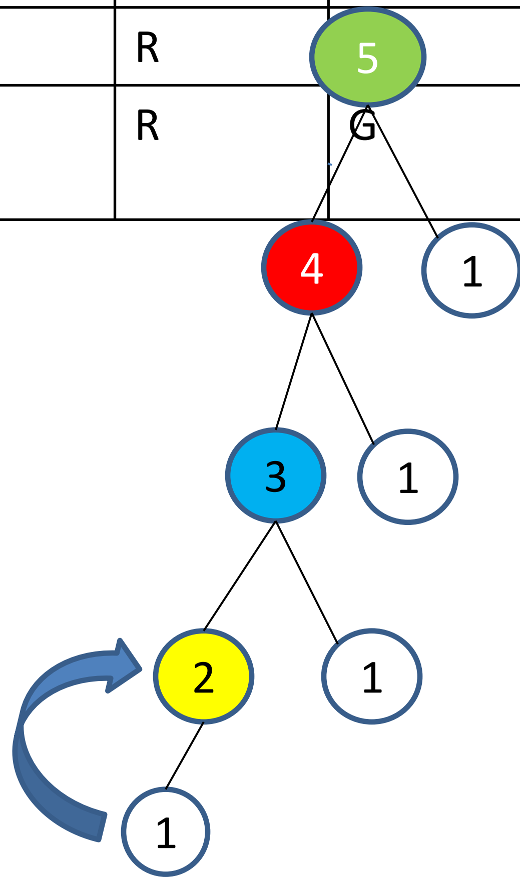
{ 5 ≠ R ,

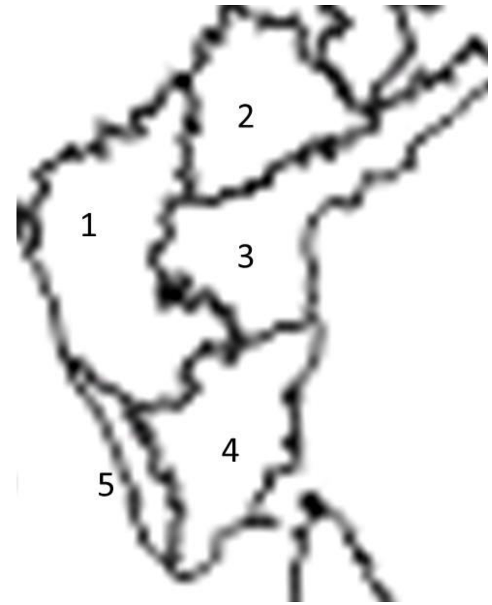
1` ≠ 2`, 1` ≠ 3`, 1` ≠ 4` , 1` ≠ 5`,

2` ≠ 3` , 3` ≠ 4`, 4` ≠ 5`}

Constraint Satisfaction Problem

1	2	3	4	5
				G
			R	G
		B	R	G
	Y	B	R	
DEAD END	Y	B	R	G





Constraints =

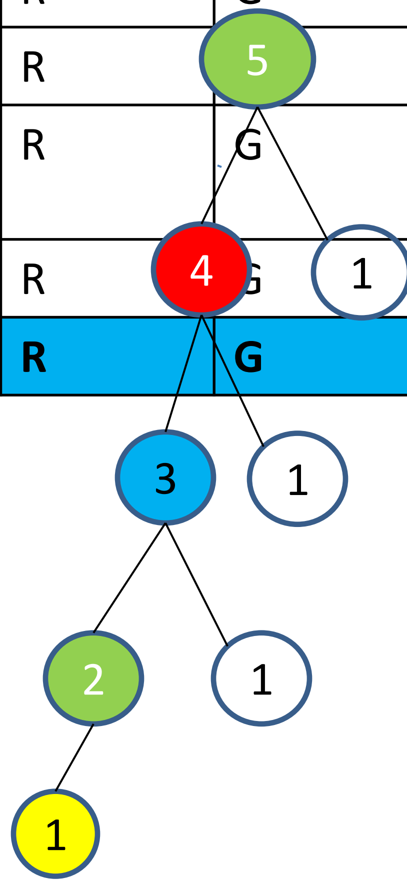
{ 5 ≠ R ,

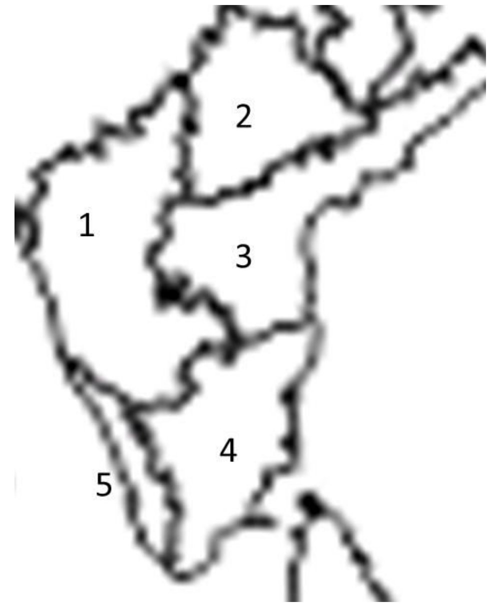
1` ≠ 2`, 1` ≠ 3`, 1` ≠ 4` , 1` ≠ 5`,

2` ≠ 3` , 3` ≠ 4`, 4` ≠ 5`}

Constraint Satisfaction Problem

1	2	3	4	5
				G
			R	G
		B	R	G
	Y	B	R	
DEAD END, BT	Y	B	R	G
	G ¥	B	R	G
Y	**G**	**B**	**R**	**G**





Constraints =

{ 5 ≠ R ,

1` ≠ 2`, 1` ≠ 3`, 1` ≠ 4` , 1` ≠ 5`,

2` ≠ 3` , 3` ≠ 4`, 4` ≠ 5`}

Search Solution to CSP

Depth-first search for CSPs with single-variable assignments is called backtracking search

```
function BACKTRACKING-SEARCH(csp) return a solution or failure
    return RECURSIVE-BACKTRACKING({} , csp)

function RECURSIVE-BACKTRACKING(assignment, csp) return a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp],assignment,csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment according to CONSTRAINTS[csp] then
            add {var=value} to assignment
            result ← RECURSIVE-BACTRACKING(assignment, csp)
            if result ≠ failure  then return result
            remove {var=value} from assignment
    return failure
```

Avenues to Improve - Heuristics

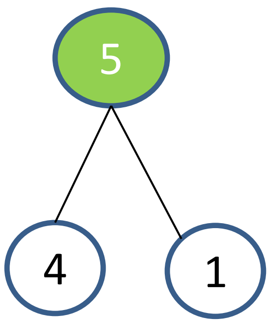
Depth-first search for CSPs with single-variable assignments is called backtracking search

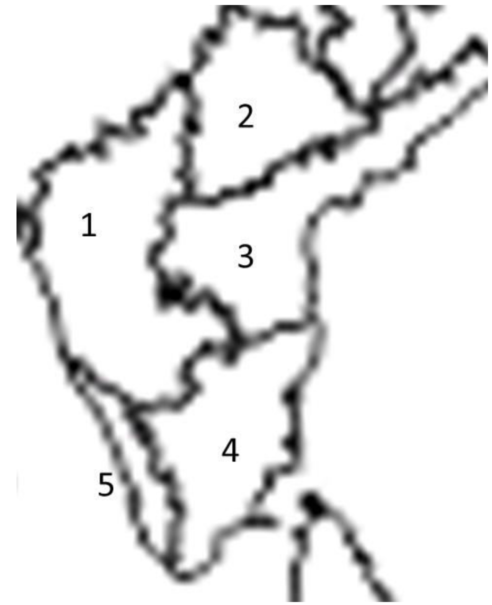
```
function BACKTRACKING-SEARCH(csp) return a solution or failure
    return RECURSIVE-BACKTRACKING({} , csp)

function RECURSIVE-BACKTRACKING(assignment, csp) return a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp],assignment,csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment according to CONSTRAINTS[csp] then
            add {var=value} to assignment
            result ← RECURSIVE-BACTRACKING(assignment, csp)
            if result ≠ failure  then return result
            remove {var=value} from assignment
    return failure
```

Constraint Satisfaction Problem

	1	2	3	4	5
	R, ~~G~~, B, Y	R, G, B, Y	R, G, B, Y	R, ~~G~~, B, Y	R, G, B, Y
R, ~~G~~, B, Y					G





Forward Checking

L1: Add a <VAR=VAL>

IF Constraint is VIOLATED

DELETE the VAL from domain

IF Assignment is not complete

IF domain is **empty** for any unassigned VAR

BACKTRACK

Else

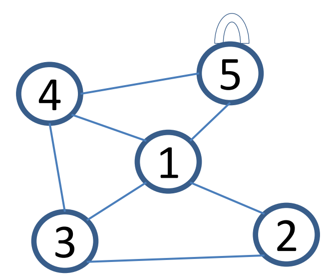
Go to L1

Constraints =

{ 5 $\neq$ R ,

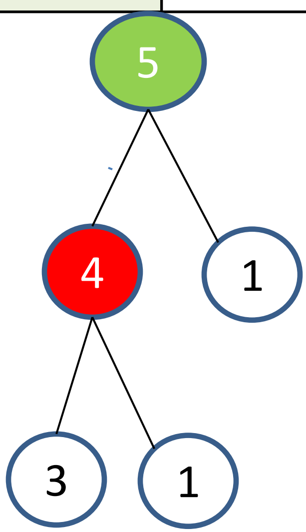
1` $\neq$ 2`, 1` $\neq$ 3`, 1` $\neq$ 4` , 1` $\neq$ 5`,

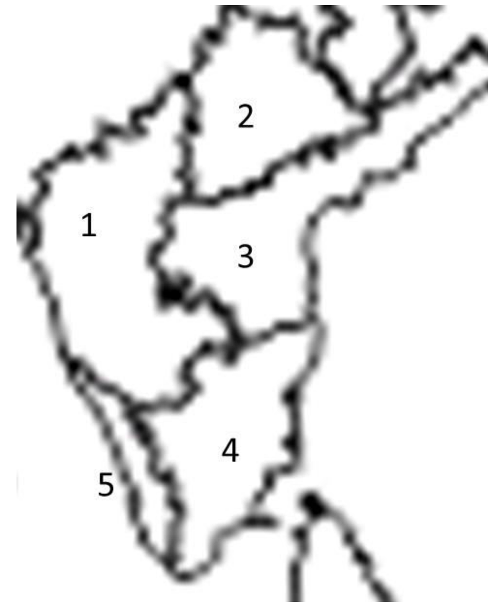
2` $\neq$ 3` , 3` $\neq$ 4`, 4` $\neq$ 5`}



Constraint Satisfaction Problem

	1	**2**	**3**	**4**	**5**
	~~R, G~~, B, Y	R, G, B, Y	~~R,~~ G, B, Y	R, ~~G,~~ B, Y	~~R,~~ G, B, Y
R, ~~G~~, B, Y					G
~~R, G.~~ B, Y				R	G





Constraints =

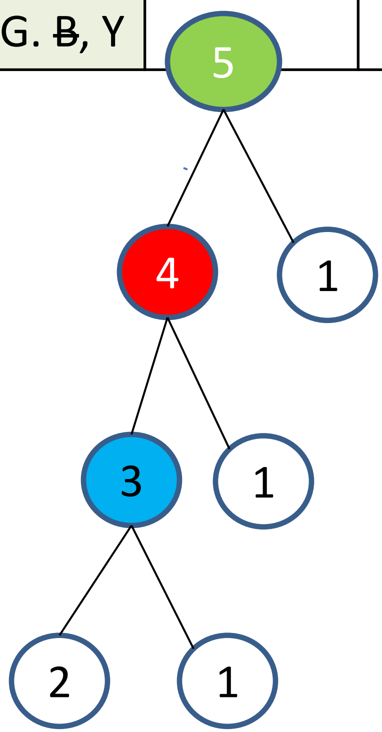
{ 5 ≠ R ,

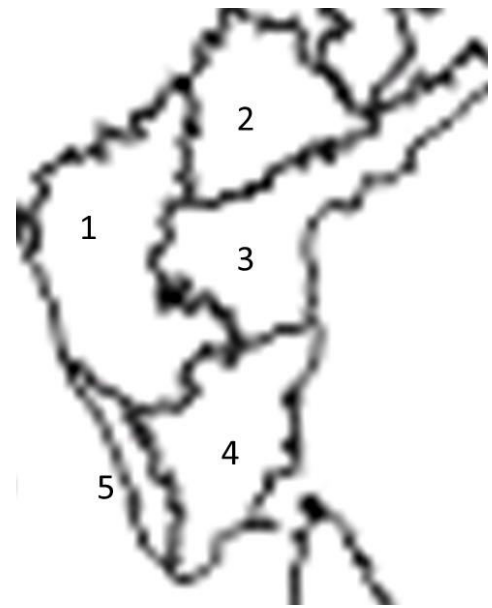
1` ≠ 2`, 1` ≠ 3`, 1` ≠ 4` , 1` ≠ 5`,

2` ≠ 3` , 3` ≠ 4`, 4` ≠ 5`}

Constraint Satisfaction Problem

	1	2	3	4	5
	~~R, G, B,~~ Y	R, G, ~~B,~~ Y	~~R,~~ G, B, Y	R, ~~G, B,~~ Y	~~R,~~ G, B, Y
R, ~~G~~, B, Y					G
~~R, G.~~ B, Y				R	G
~~R~~, G. ~~B~~, Y			B	R	G





Constraints =

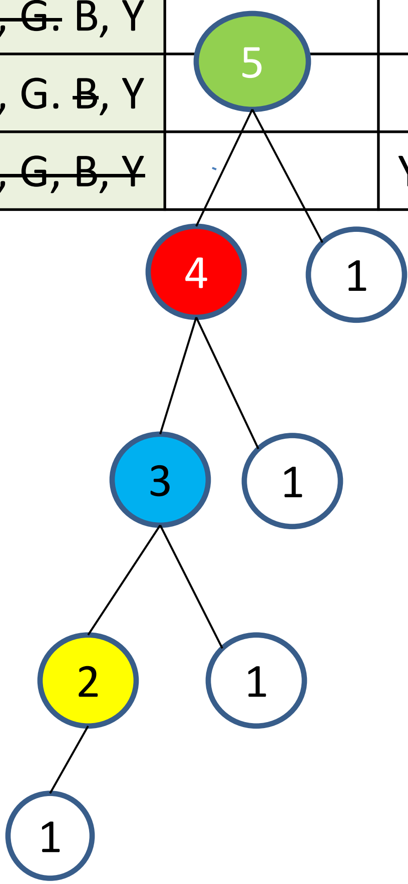
{ 5 ≠ R ,

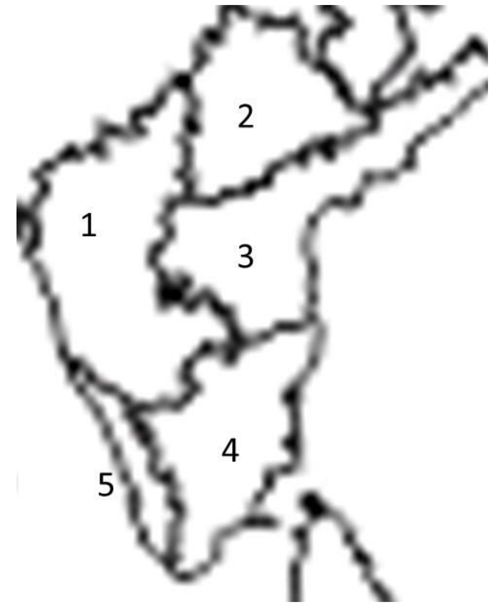
1` ≠ 2`, 1` ≠ 3`, 1` ≠ 4` , 1` ≠ 5`,

2` ≠ 3` , 3` ≠ 4`, 4` ≠ 5`}

Constraint Satisfaction Problem

	1	2	3	4	5
	~~R, G, B, Y~~	R, G~~, B,~~ Y	~~R,~~ G, B~~, Y~~	R~~, G, B,~~ Y	~~R~~, G, B, Y
R, ~~G~~, B, Y					G
~~R, G.~~ B, Y				R	G
~~R~~, G. ~~B~~, Y			B	R	G
~~R, G, B, Y~~		Y	B	R	G





Constraints =

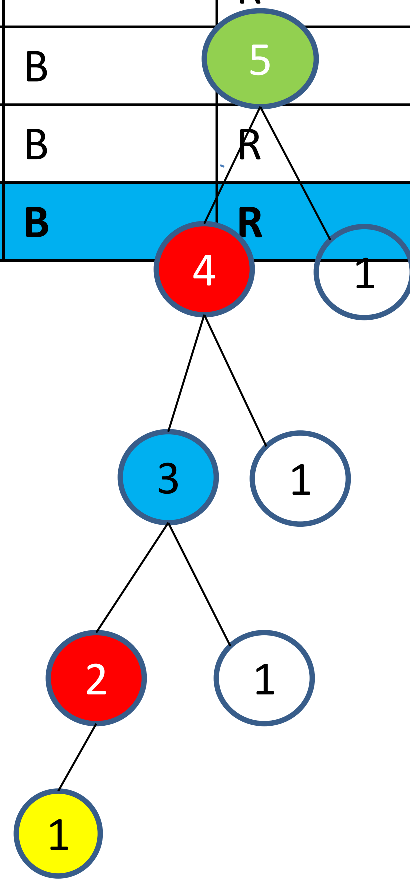
{ 5 $\neq$ R ,

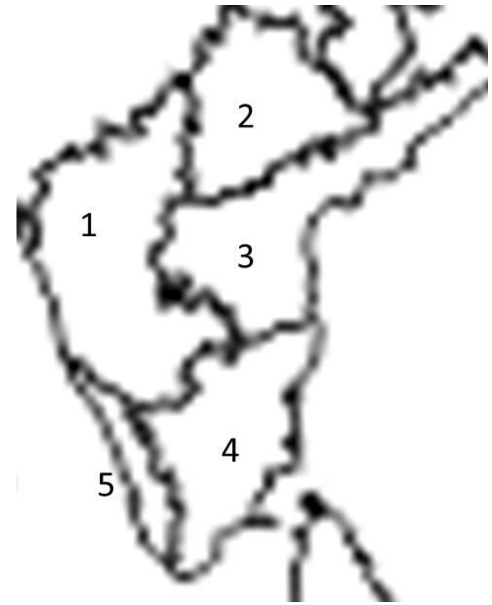
1` $\neq$ 2`, 1` $\neq$ 3`, 1` $\neq$ 4` , 1` $\neq$ 5`,

2` $\neq$ 3` , 3` $\neq$ 4`, 4` $\neq$ 5`}

Constraint Satisfaction Problem

	1	**2**	**3**	**4**	**5**
	~~R, G, B~~, Y	R, ~~G, B~~, Y	~~R~~, G, B, ~~Y~~	R, ~~G, B~~, Y	~~R~~, G, B, Y
R, ~~G~~, B, Y					G
~~R, G~~. B, Y				R	G
~~R~~, G. ~~B~~, Y			B		G
~~R, G, B~~, Y		R	B	R	G
	Y	**R**	**B**	**R**	**G**





Constraints =

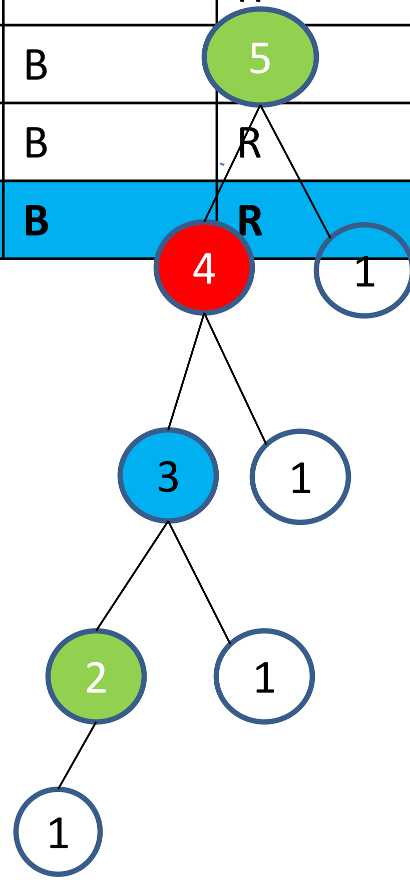
{ 5 ≠ R ,

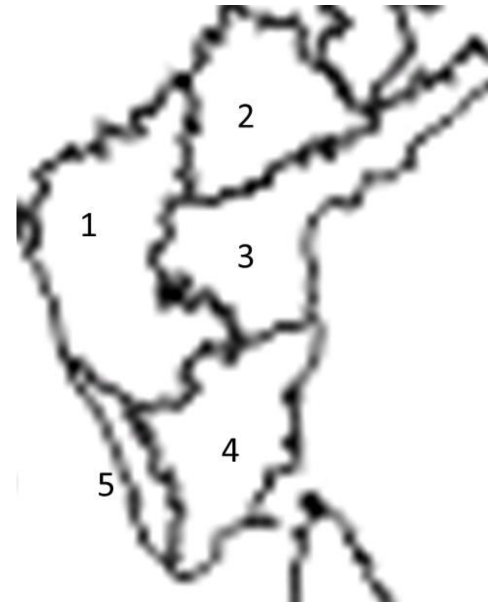
1\` ≠ 2\`, 1\` ≠ 3\`, 1\` ≠ 4\` , 1\` ≠ 5\`,

2\` ≠ 3\` , 3\` ≠ 4\`, 4\` ≠ 5\`}

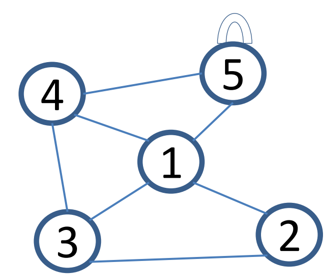
Constraint Satisfaction Problem - Graph

	1	**2**	**3**	**4**	**5**
	~~R, G, B~~, Y	R, G, ~~B~~, Y	~~R, G~~, B, Y	R, ~~G, B~~, Y	~~R~~, G, B, Y
R, ~~G~~, B, Y					G
~~R, G.~~ B, Y				R	G
~~R~~, G. ~~B~~, Y			B		G
~~R, G, B,~~ Y		G???	B	R	G
	Y	**R**	**B**	**R**	**G**





Constraints =



{5 ≠ R ,

1\` ≠ 2\`, 1\` ≠ 3\`, 1\` ≠ 4\` , 1\` ≠ 5\`,

2\` ≠ 3\` , 3\` ≠ 4\`, 4\` ≠ 5\`}

Heuristics

- Frequent Techniques
  - Most constrained variable
  - Most constraining variable
  - Least constraining value
  - Forward checking

- MRV / Most constrained variable
  - choose the variable with the fewest legal values

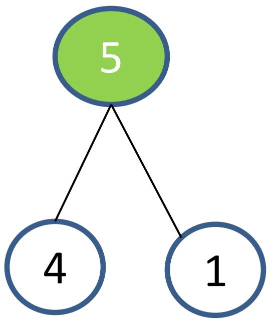
- MCV / Most constraining variable
  - choose the variable with the most constraints on remaining variables

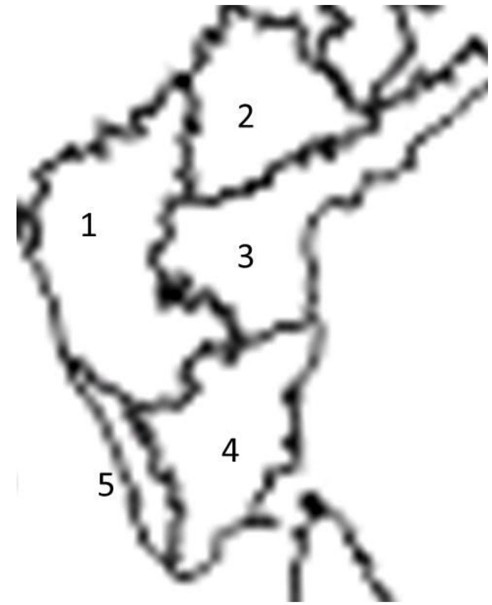
- LCV/ Least constraining value
  - Choose value that rules out the fewest values in the remaining variables

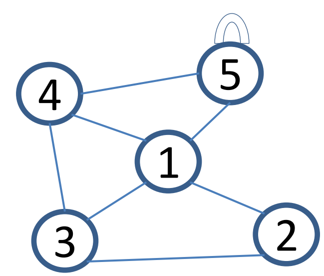
- Forward checking
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

Constraint Satisfaction Problem

1	2	3	4	5
R, ~~G~~, B, Y	R, G, B, Y	R, G, B, Y	R, ~~G~~, B, Y	R, G, B, Y
				G







Constraints =

{ 5 ≠ R ,

1` ≠ 2`, 1` ≠ 3`, 1` ≠ 4` , 1` ≠ 5`,

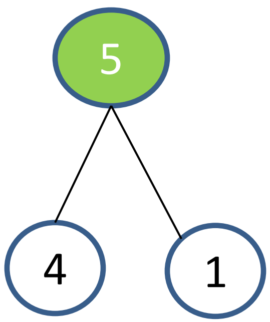
2` ≠ 3` , 3` ≠ 4`, 4` ≠ 5`}

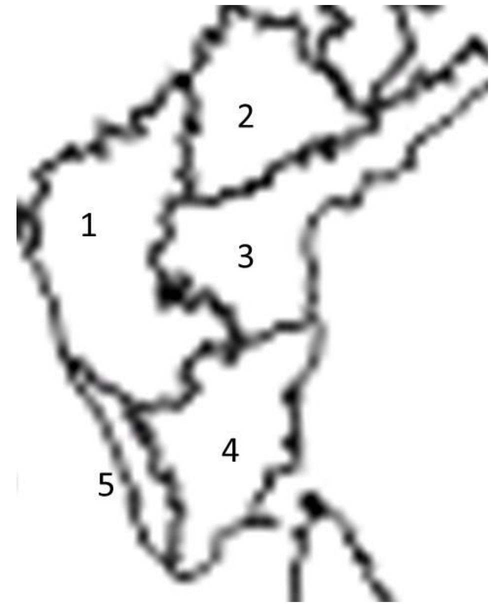
H1: MRV / Most constrained variable

Choose the variable with the fewest legal values

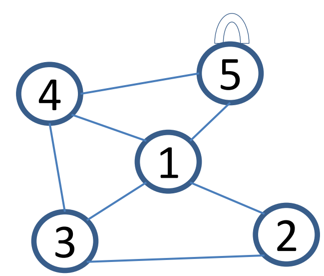
Constraint Satisfaction Problem

1	2	3	4	5
R, ~~G~~, B, Y	R, G, B, Y	R, G, B, Y	R, ~~G~~, B, Y	R, G, B, Y
				G





Constraints =



{5 ≠ R ,

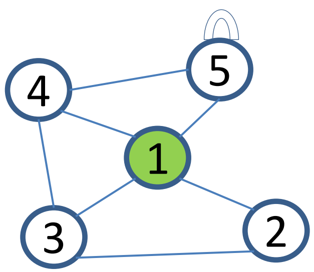
1` ≠ 2`, 1` ≠ 3`, 1` ≠ 4` , 1` ≠ 5`,

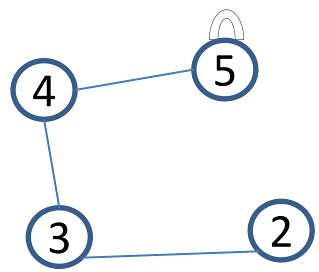
2` ≠ 3` , 3` ≠ 4`, 4` ≠ 5`}

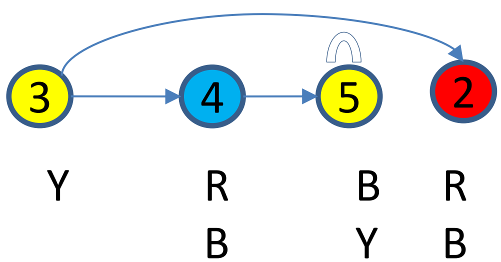
H2: MCV / Most constraining variable

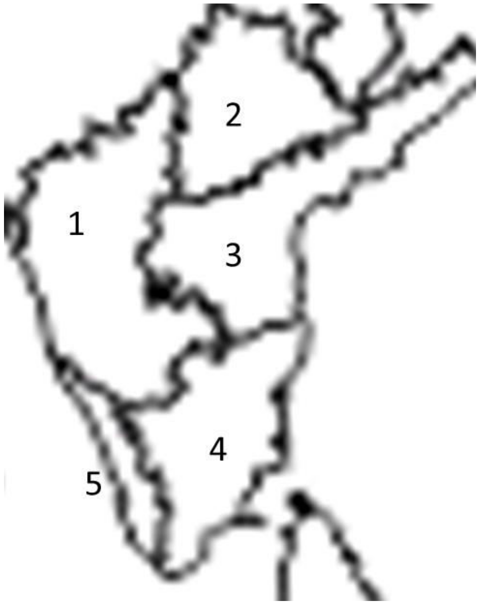
Choose the variable with the most constraints on remaining variables

Constraint Satisfaction Problem









Constraints =

{ 5 ≠ R ,

1` ≠ 2`, 1` ≠ 3`, 1` ≠ 4` , 1` ≠ 5`,

2` ≠ 3` , 3` ≠ 4`, 4` ≠ 5`}

Heuristics

- Frequent Techniques
  - Most constrained variable
  - Most constraining variable
  - Least constraining value
  - Forward checking

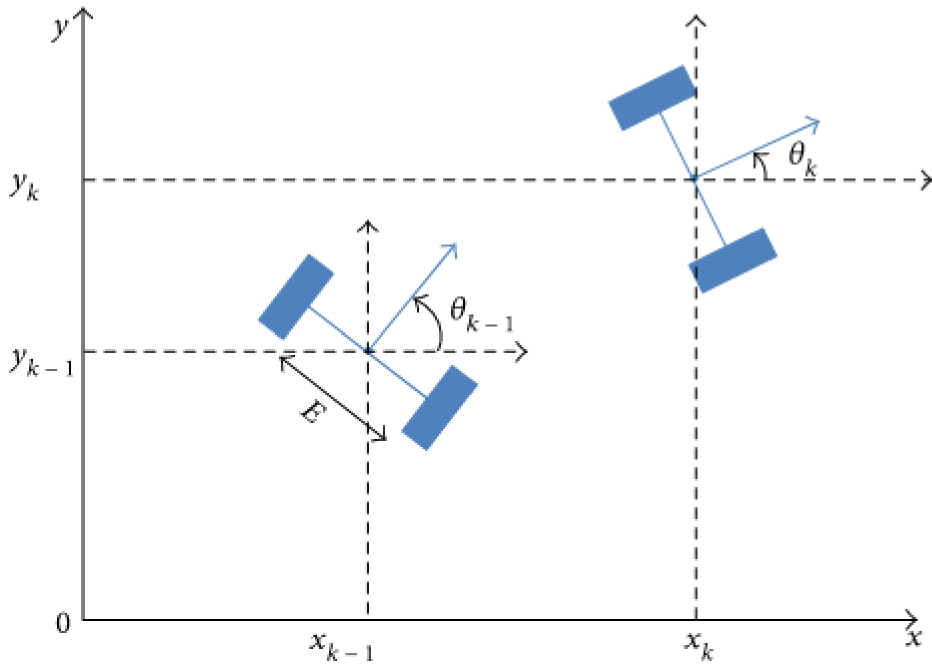
- MRV / Most constrained variable
  - choose the variable with the fewest legal values

- MCV / Most constraining variable
  - choose the variable with the most constraints on remaining variables

- LCV/ Least constraining value
  - Choose value that rules out the fewest values in the remaining variables

- Forward checking
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

CSP in Vehicle Localization



Source Credit:

[2018 -Localization of a Vehicle: A Dynamic Interval Constraint Satisfaction Problem-Based Approach]()

Local Search for CSP

Sudoku Problem

Constraint Satisfaction Graph - Subgraph

Sudoku

	1	2	3	4
A		2	1	
B	4			
C				2
D		3		

	1	2	3	4
A	3	2	1	3
B	4	2	3	4
C	2	4	4	2
D	1	3	2	3

	1	2	3	4
A	3	2	1	3
B	4	2	3	4
C	2	4	4	2
D	1	3	1	3

	1	2	3	4
A	3	2	1	3
B	4	2	3	4
C	2	4	4	2
D	1	3	2	3

	1	2	3	4
A	3	2	1	3
B	4	2	3	4
C	2	4	4	2
D	1	3	3	3

	1	2	3	4
A	3	2	1	3
B	4	2	3	4
C	2	4	4	2
D	1	3	4	3

Next Class

CSP as Local Search Problem (Sudoku)

CSP AC-3 algorithm (Map Coloring Problem)

Inference & Reasoning using Logic (Shared few materials to refresh the concept of propositional & predicate logic & resolution methods over canvas page. Please refresh these before the next class)

Required Reading: AIMA - Chapter #5.1, #5.2, #5.3, #5.4

Thank You for all your Attention

Note : Some of the slides are adopted from AIMA TB materials