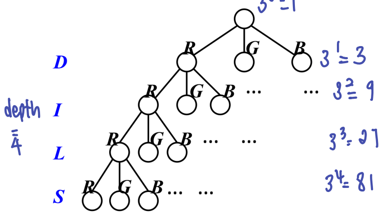
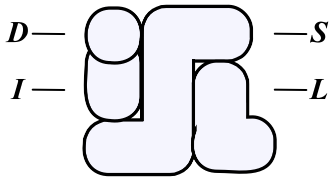
Tutorial 2

2.1 The logo of the Intelligent Systems Laboratory comprises four design elements: the letters *I*, *S*, *L*, and the circular dot *D*. The graphic designer now wants to color each element red, green, or blue, such that adjacent elements sport different colors.

(No forward check)
DFS + Backtracking



Branching Factor = 3
Total # of Nodes = 1+3+27 . . .
= 121

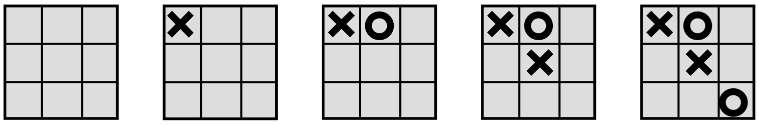


You are asked to use a *Depth-First Search* algorithm *with Forward Checking* to generate all possible color combinations for the logo. In both cases below, draw the complete search space, calculate the average branching factor and the total number of nodes, and comment on the efficiency of the approach.

(a) Assume the algorithm arbitrarily colors the elements in *alphabetical order*.

(b) Assume the algorithm colors the elements in the *order indicated by the best heuristics* that can be used for Constraint Satisfaction Problems. Explain why it is the best in this case and how useful the others are (or not).

2.2 The game of Tic-Tac-Toe is played between two players on a board composed of 3x3 squares. Each player takes turn placing an X or O in one empty square, as illustrated by the sample game below (first 4 moves). The X or O player wins the game by aligning 3 X's or O's, respectively, on the same column, row, or diagonal.



A *heuristics* for this game is proposed as follows. Let X_n be the number of rows, columns, or diagonals with exactly *n* X's and no O's. Similarly, let O_n be the number rows, columns, or diagonals with *n* O's and no X's. The heuristics gives any non-terminal position a value equal to: $3\,X_2 + X_1 - 3\,O_2 - O_1$. Finally, the heuristics assigns +1 to positions with X_3=1 (a win for X), –1 to positions with O_3=1 (a loss for X), and 0 to all other terminal positions (a draw).

(a) Show the *game tree* up to depth 2 i.e., with one X and one O on the board. Make sure to take *symmetry* into account to minimise the size of the tree.

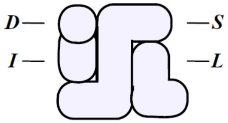
(b) Indicate on the game tree the *estimated values* of all positions at depth 2. Use the *MiniMax* algorithm to derive the backed-up values for positions at depths 1 and 0. Determine then the best starting move for the X player.

CZ3005 Tutorial 2

Sinno Jialin PAN
Email: sinnopan@ntu.edu.sg
Homepage: http://www.ntu.edu.sg/home/sinnopan/index.html

Q1: ISL Logo Coloring Problem

The logo of the Intelligent Systems Laboratory comprises four design elements: the letters *I*, *S*, *L*, and the circular dot *D*. The graphic designer now wants to color each element red, green, or blue, such that adjacent elements sport different colors.



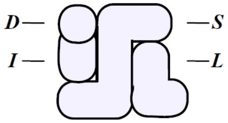
You are asked to use a *Depth-First Search* algorithm *with Forward Checking* to generate all possible color combinations for the logo. In both cases below, draw the complete search space, calculate the average branching factor and the total number of nodes, and comment on the efficiency of the approach.

(a) Assume the algorithm arbitrarily colors the elements in *alphabetical order*.

(b) Assume the algorithm colors the elements in the *order indicated by the best heuristics* that can be used for Constraint Satisfaction Problems. Explain why it is the best in this case and how useful the others are (or not).

Q1 (a)

The logo of the Intelligent Systems Laboratory comprises four design elements: the letters *I*, *S*, *L*, and the circular dot *D*. The graphic designer now wants to color each element red, green, or blue, such that adjacent elements sport different colors.

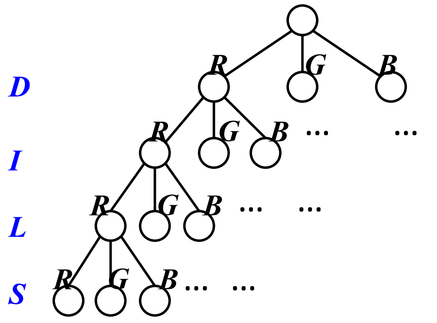


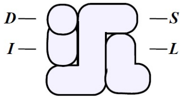
You are asked to use a *Depth-First Search* algorithm *with Forward Checking* to generate all possible color combinations for the logo. In both cases below, draw the complete search space, calculate the average branching factor and the total number of nodes, and comment on the efficiency of the approach.

(a) Assume the algorithm arbitrarily colors the elements in *alphabetical order*.

(b) Assume the algorithm colors the elements in the *order indicated by the best heuristics* that can be used for Constraint Satisfaction Problems. Explain why it is the best in this case and how useful the others are (or not).

DFS (no Forward Checking)





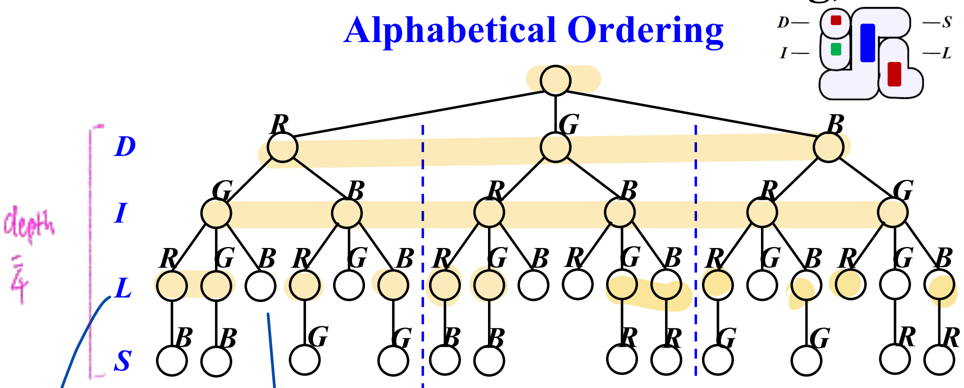
DFS a good choice?

- Yes: depth-limited search space, all solutions at maximum depth, many possible solutions
- **No**: need to generate all paths, blind search generates invalid colorings

- Depth: 4
- Branching factor: 3
- Number of nodes:
  $1 + 3 + 3^2 + 3^3 + 3^4 = 121$

DFS with Forward Checking,

Alphabetical Ordering



Nodes at each level

1

3

2x3 = 6

3x6 = 18

2x6 = 12

∴ total = 40

- Number of nodes: 1 + 3 + 3×2 + 6×3 + 6×2 = 40
- Avg. branching factor: $\frac{(40-1)}{1+3+3\times2+6\times2} = 1.77$

Formula = $\frac{\text{\# Non root nodes}}{\text{\# Non Leaf Nodes.}}$

only neighbour with S & no color yet ∴ can be all colours

S is neighbouring to all 3 letters ∴ cannot expand anymore.

# of nodes expanded / non-leaf

Root node

PROS: High Efficiency, no invalid colouring

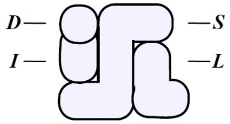
CONS: Elements are assigned colour in arbitrary (alphabetical) order ∴ Not optimal (didn't start with S involving the largest # of constrains)

General CSP Heuristics

- Most Constraining Variable (Degree): select among yet unassigned variables the one involved in the largest number of constraints
  - i.e. logo element with the largest number of neighbors
    - useful to optimize the order of variable assignments
- Least Constraining Value: select a value that leaves the largest choice of values for other constraint-related variables
  - i.e., the color that less constrains other logo elements
    - useful to prevent deadlocks, reduce backtracking
- Most Constrained Variable (Minimum-Remaining Value): select the variable with fewest possible values
  - i.e., logo element with fewest colors available
    - useful to complement the first heuristic in case of a tie

Q1 (b)

The logo of the Intelligent Systems Laboratory comprises four design elements: the letters *I*, *S*, *L*, and the circular dot *D*. The graphic designer now wants to color each element red, green, or blue, such that adjacent elements sport different colors.



You are asked to use a *Depth-First Search* algorithm *with Forward Checking* to generate all possible color combinations for the logo. In both cases below, draw the complete search space, calculate the average branching factor and the total number of nodes, and comment on the efficiency of the approach.

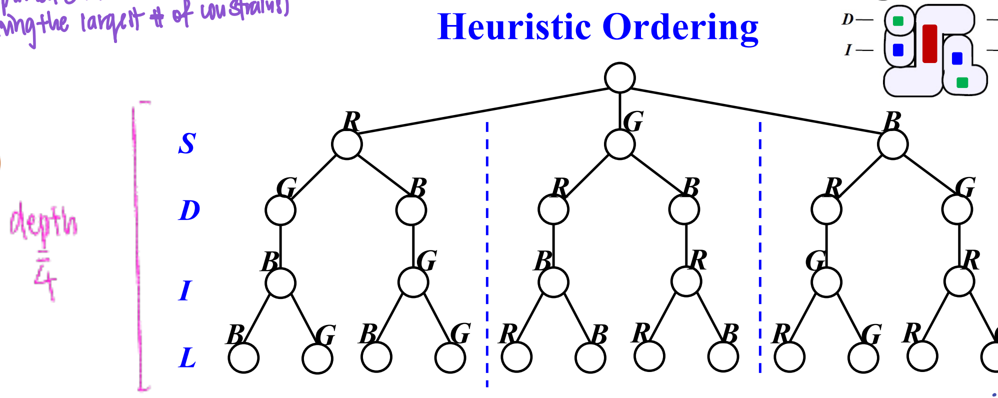
(a) Assume the algorithm arbitrarily colors the elements in *alphabetical order*.

(b) Assume the algorithm colors the elements in the *order indicated by the best heuristics* that can be used for Constraint Satisfaction Problems. Explain why it is the best in this case and how useful the others are (or not).

☆ most constraining element: S = 3 Neigh ✓
D = 2 Neigh
L = 1 Neigh

DFS with Forward Checking,

Heuristic Ordering



Noded at each level

1

3

2x3 = 6

1x6 = 6

2x6 = 12

∴ total = 28

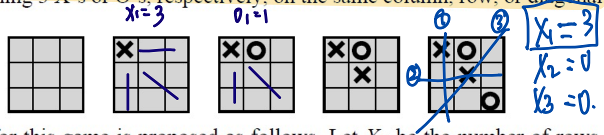
- Number of nodes: 1 + 3 + 3×2 + 6×1 + 6×2 = 28
- Avg. branching factor: $\frac{(28-1)}{1+3+3\times2+6\times1} = 1.69$

$\frac{\text{\# Not root nodes}}{\text{\# Non leaf nodes.}}$

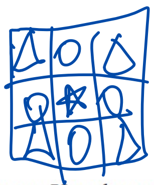
PROs: High Efficiency, no invalid colouring, no backtracking, elements are assigned colours in optimal order.

Tic-Tac-Toe Game Tree

4.2 The game of Tic-Tac-Toe is played between two players on a board composed of 3x3 squares. Each player takes turn placing an X or O in one empty square, as illustrated by the sample game below (first 4 moves). The X or O player wins the game by aligning 3 X's or O's, respectively, on the same column, row, or diagonal.



A *heuristics* for this game is proposed as follows. Let X_n be the number of rows, columns, or diagonals with exactly n X's and no O's. Similarly, let O_n be the number rows, columns, or diagonals with n O's and no X's. The heuristics gives any non-terminal position a value equal to: $3X_2 + X_1 - 3O_2 - O_1$. Finally, the heuristics assigns +1 to positions with X_3=1 (a win for X), –1 to positions with O_3=1 (a loss for X), and 0 to all other terminal positions (a draw).

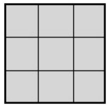


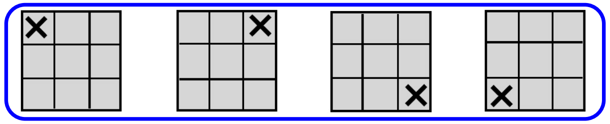
symmetric

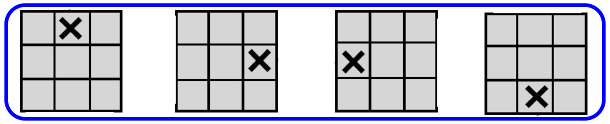
3 choices only instead of 9 choices.

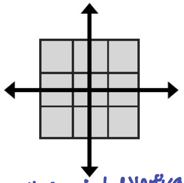
(a) Show the *game tree* up to depth 2 i.e., with one X and one O on the board. Make sure to take *symmetry* into account to minimise the size of the tree.

- Horizontal and vertical symmetry









Horizontal & Vertical symmetry

- Only 3 moves at depth 1 (v.s 9)
- Only 12 moves at depth 2 (v.s 72)

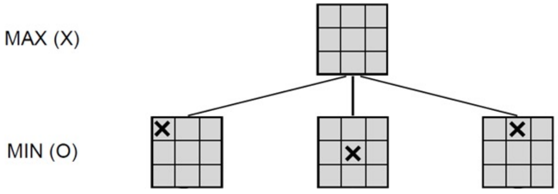
Minimax Search Strategy

- **Minimax search strategy**
  - Maximize one's own utility and minimize the opponent's
    - Assumption is that the opponent does the same
- **3-step process:** **Impractical**
  1. Generate the entire game tree down to terminal states
  2. Calculate utility
     1. Assess the utility of each terminal state
     2. Determine the best utility of the parents of the terminal state
     3. Repeat the process for their parents until the root is reached
  3. Select the best move (i.e. with the highest utility value)

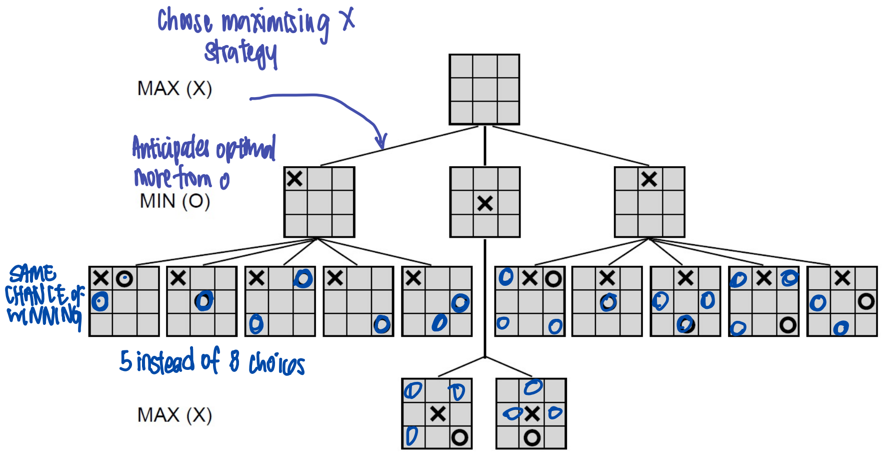
Replace utility function by evaluation function based on heuristics

Estimate of the expected utility

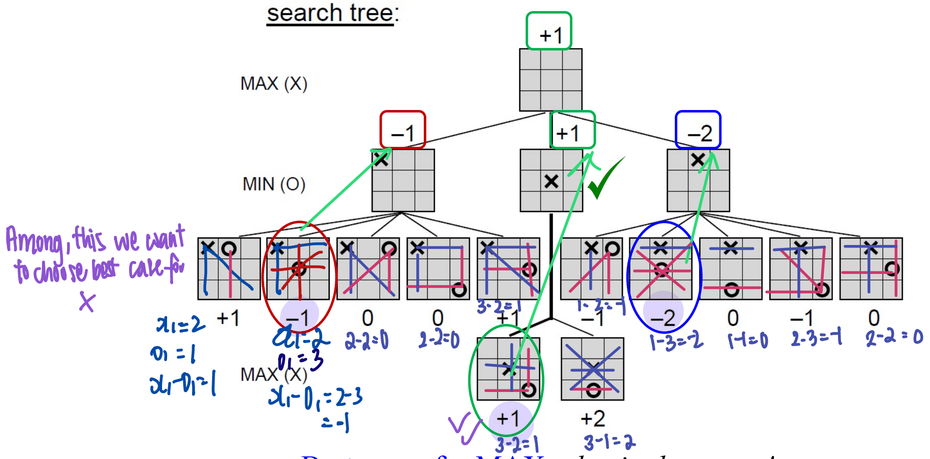
Game Tree without Heuristics



Game Tree without Heuristics (cont.)



Game Tree with Heuristics



Best move for MAX: *play in the centre!*

↳ Need to use min & max algorithm.

(b) Indicate on the game tree the *estimated values* of all positions at depth 2. Use the *MiniMax* algorithm to derive the backed-up values for positions at depths 1 and 0. Determine then the best starting move for the X player.

- Value of non–terminal nodes:

$$h = 3X_2 + X_1 - 3O_2 - O_1$$

i.e., $h = 3(X_2 - O_2) + (X_1 - O_1)$

- At depth 2: $X_2 = O_2 = 0$

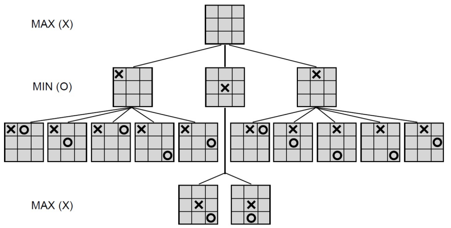
$$h = X_1 - O_1$$

since only consider first 2 moves

$x_2 = O_2 = 0$

x & O only move only 1 time.

there will be no 2Os & 2 Xs placed at depth 2


MAX (X)
MIN (O)
MAX (X)