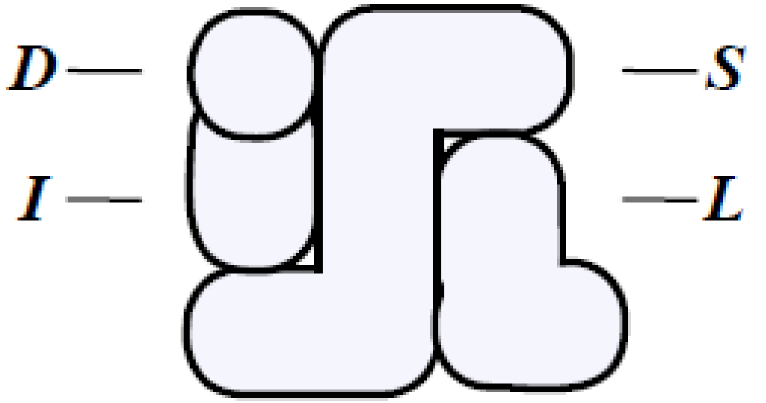
Artificial Intelligence

CSP Game Playing

Lu Shijian

Assoc Prof, SCSE

Shijian.Lu@ntu.edu.sg
N4-02C-101



Problem01

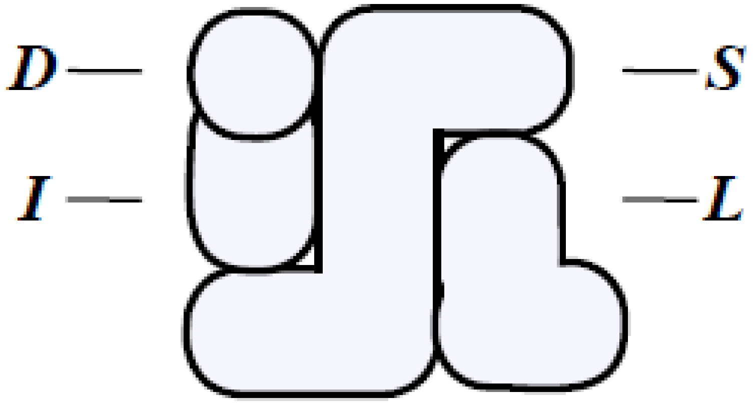
The logo of the Intelligent Systems Laboratory comprises four design elements: the letters *I*, *S*, *L*, and the circular dot *D.* The graphic designer now wants to color each element red, green, or blue, such that adjacent elements sport different colours.

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.

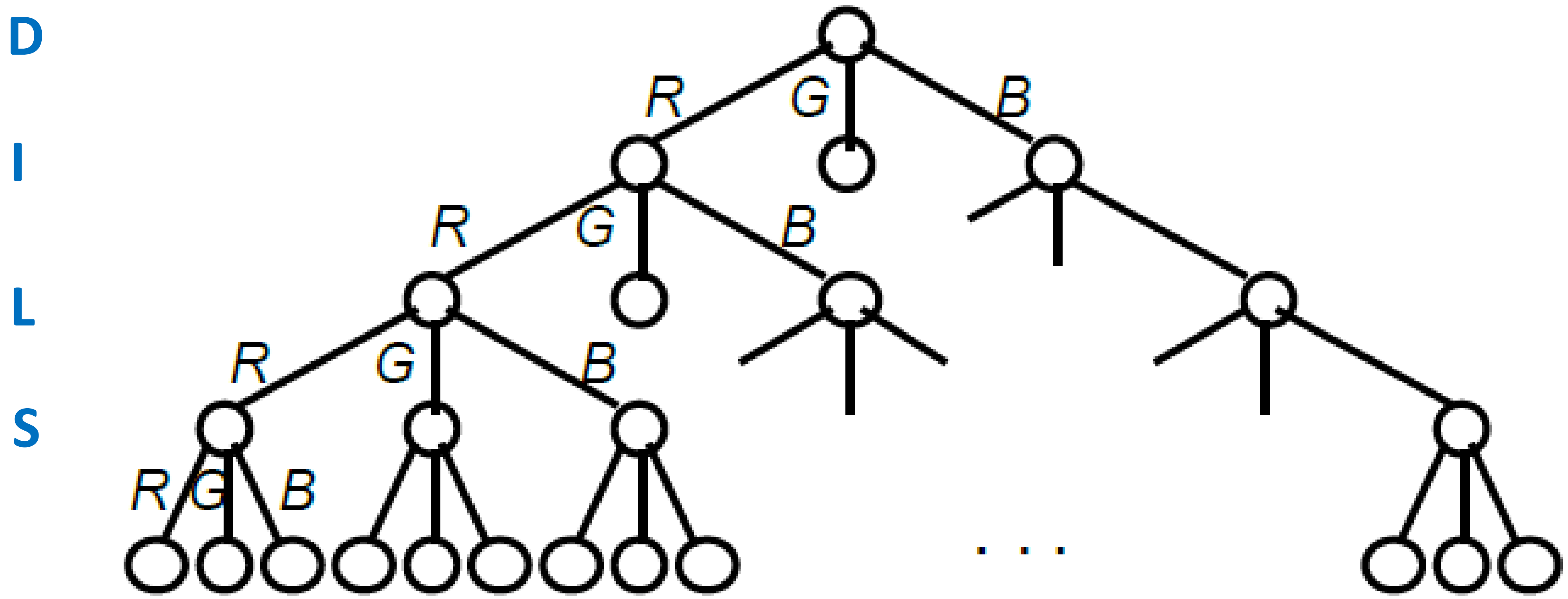
a) Assume the algorithm arbitrarily colours the elements in *alphabetical order*.

b) Assume the algorithm colours 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).

Problem01



Depth-first search w/o forward checking: arbitrary colouring elements in *alphabetical order*.



Problem01

Uniform depth of 4

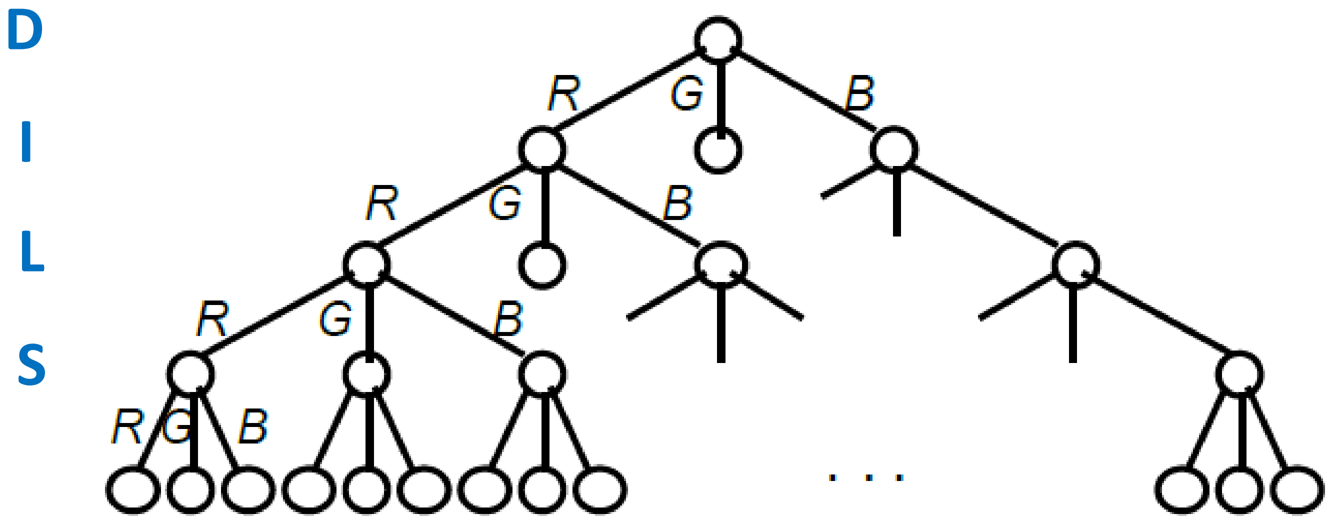
branching factor = 3

Number of nodes: exactly, $1 + 3 + 3^2 + 3^3 + 3^4 \ = 121$

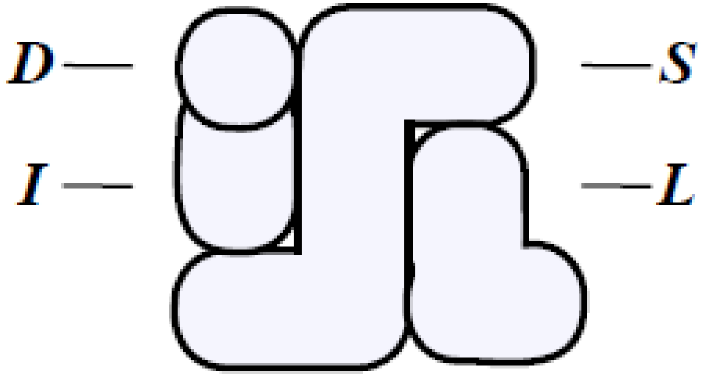
DFS a good choice?

yes → depth-limited search space, all solutions at maximum depth, many possible solutions

no → need to generate all solutions, blind search generates invalid colourings

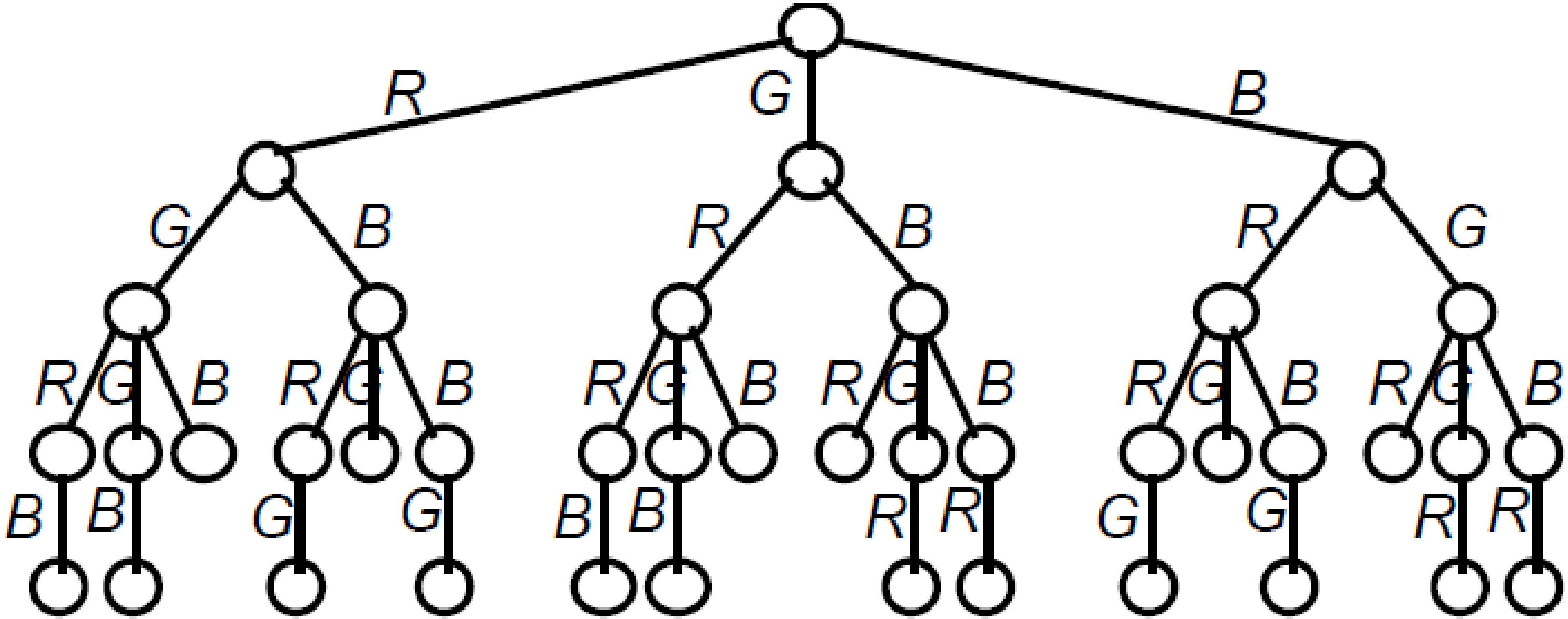


Problem01

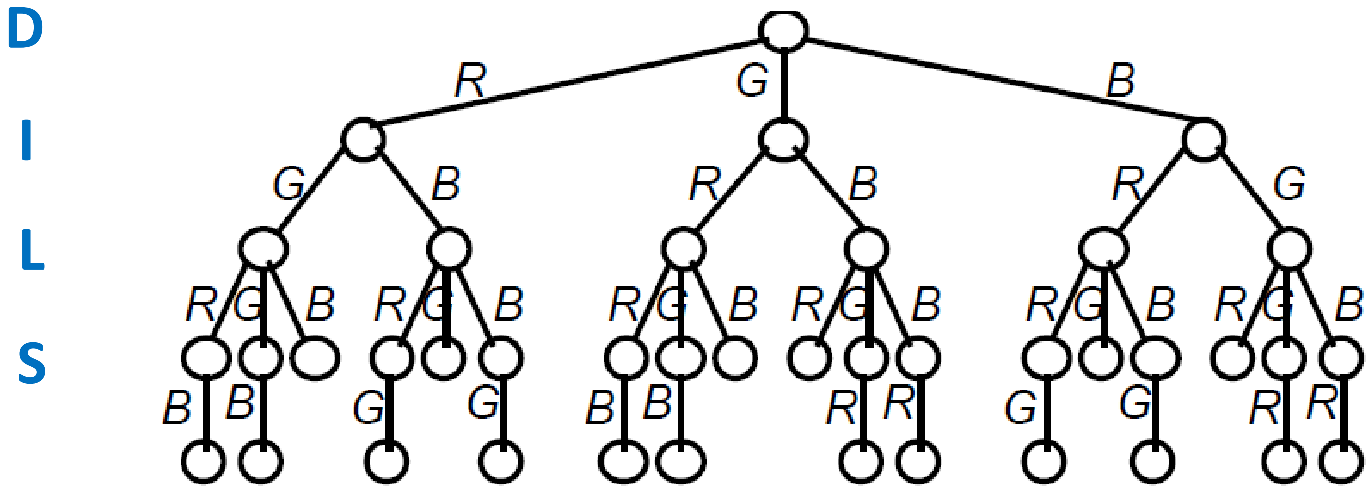


Depth-first search with forward checking and arbitrary ordering - forward checking takes constraints into account to significantly prune the search space

D
I
L
S



Problem01

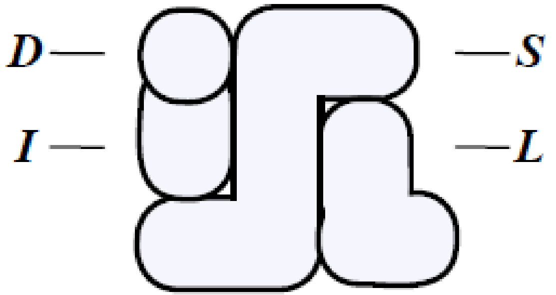


Number of nodes: 1 + 3 + 3∗2 + 3*6 + 2*6 = 40

Average branching factor: (40 – 1) / (1 + 3 + 6 + 6∗2) = 1.77

Efficiency: significant improvement i.e., search space reduced from 121 to 40 nodes, branching factor decreased from 3 to 1.77, no invalid colourings, less backtracking

However, elements are assigned a color in arbitrary (alphabetical) order → *not* optimal



Problem01

Depth-first search with forward checking: colouring elements by best-heuristic order

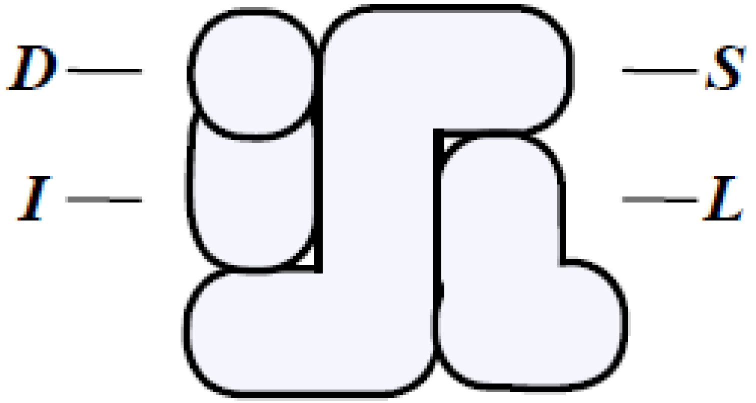
Most Constraining Variable selects among yet unassigned variables the one involved in the largest number of constraints, i.e., logo element with the largest # of neighbours

→useful to optimize the order of variable assignments

Least Constraining Value selects a value that leaves the largest choice of values for other constraint-related variables, i.e., color that less constrains other logo elements

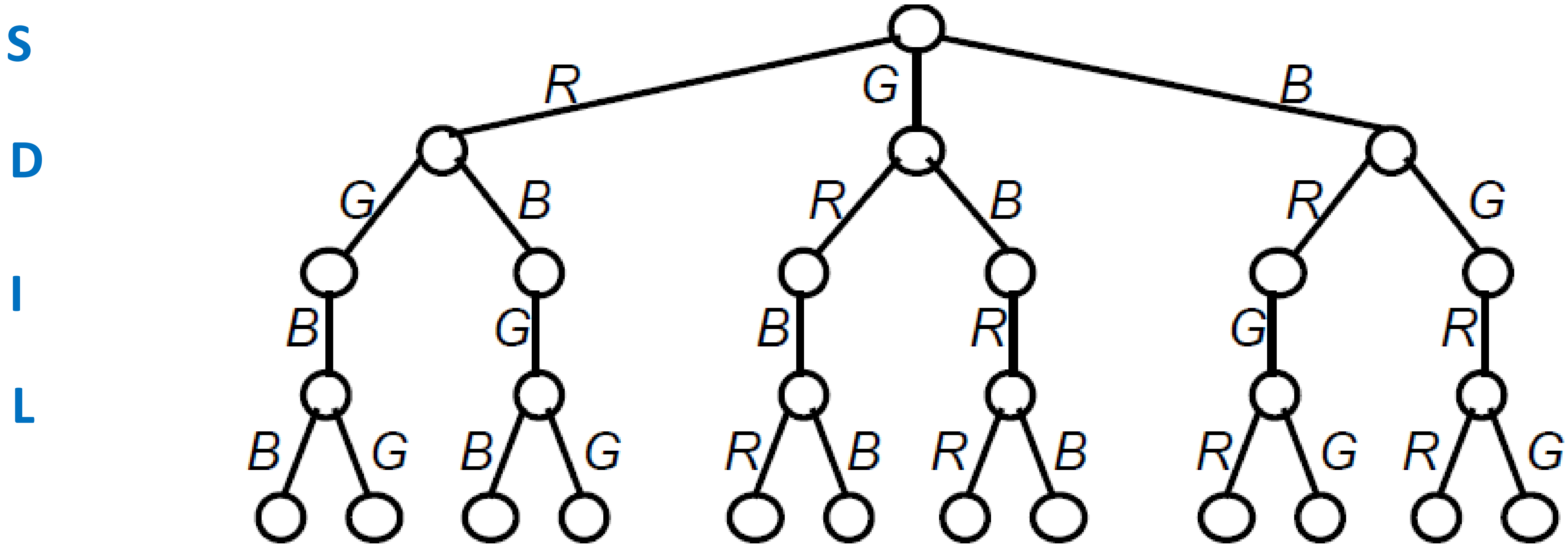
→useful to prevent deadlocks, reduce backtracking

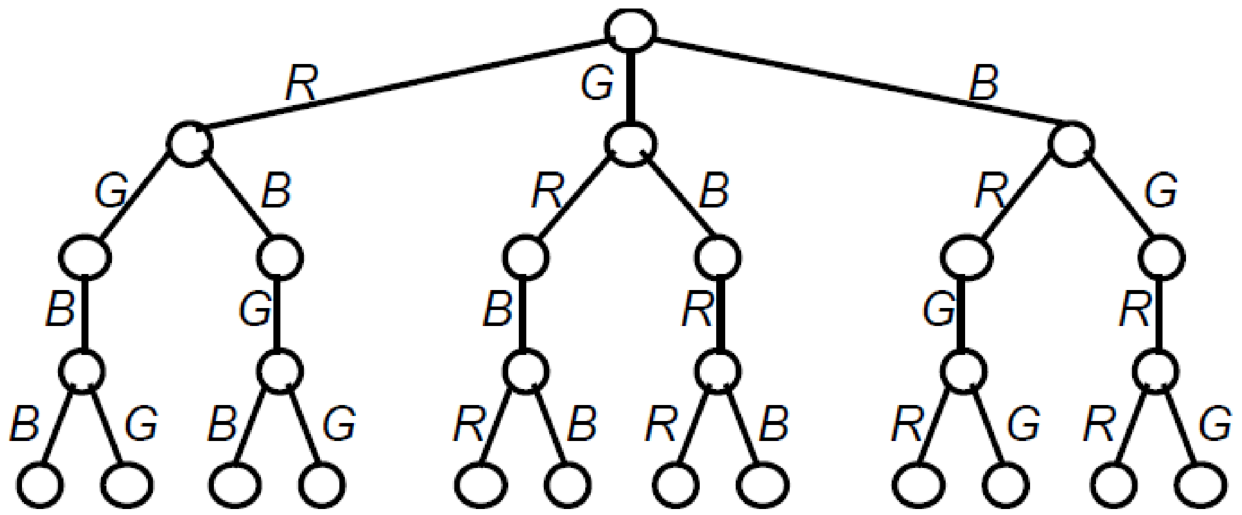
best heuristics: *Most Constraining Variable*



Problem01

Depth-first search with forward checking and best heuristic ordering: most constraining element: *S* with 3 neighbours, followed by *D* and *I* with 2 neighbours each, then *L* with only 1 neighbour





Problem01

Number of nodes: $1 + 3 + 3*2 + 6*1 + 6*2 = 28$

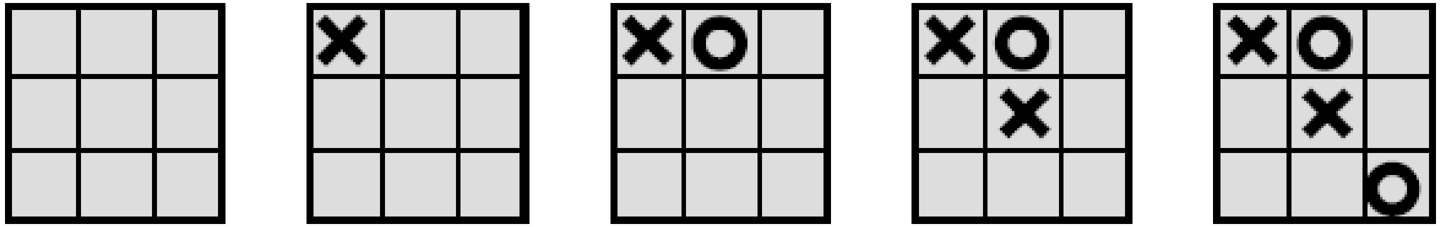
Average branching factor: $(28 – 1) / (1 + 3 + 6 + 6*1) = 1.69$

Efficiency: further good improvement i.e., search space reduced from 40 to 28 nodes, branching factor decreased from 1.77 to 1.69, no invalid colourings, no backtracking

Elements are assigned a color in *optimal* order

Problem02

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 3 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).

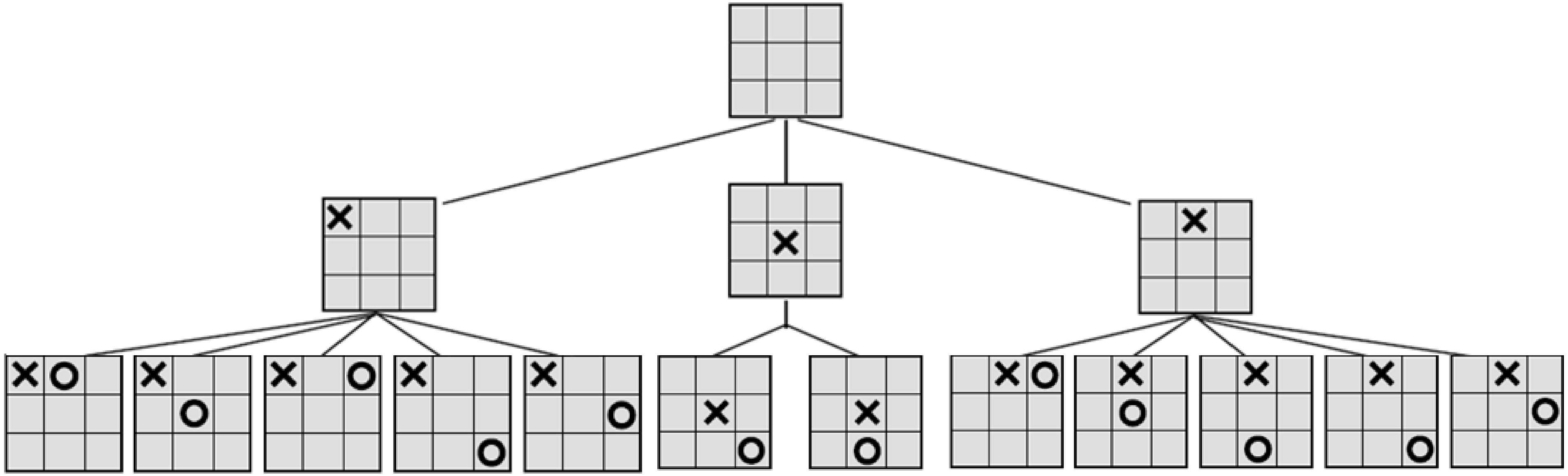
Problem02

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.

Considering the symmetry in horizontal, vertical, and diagonal directions, there are 3 moves (vs 9) at depth 1, and 12 moves (vs 72) at depth 2.

MAX(X)

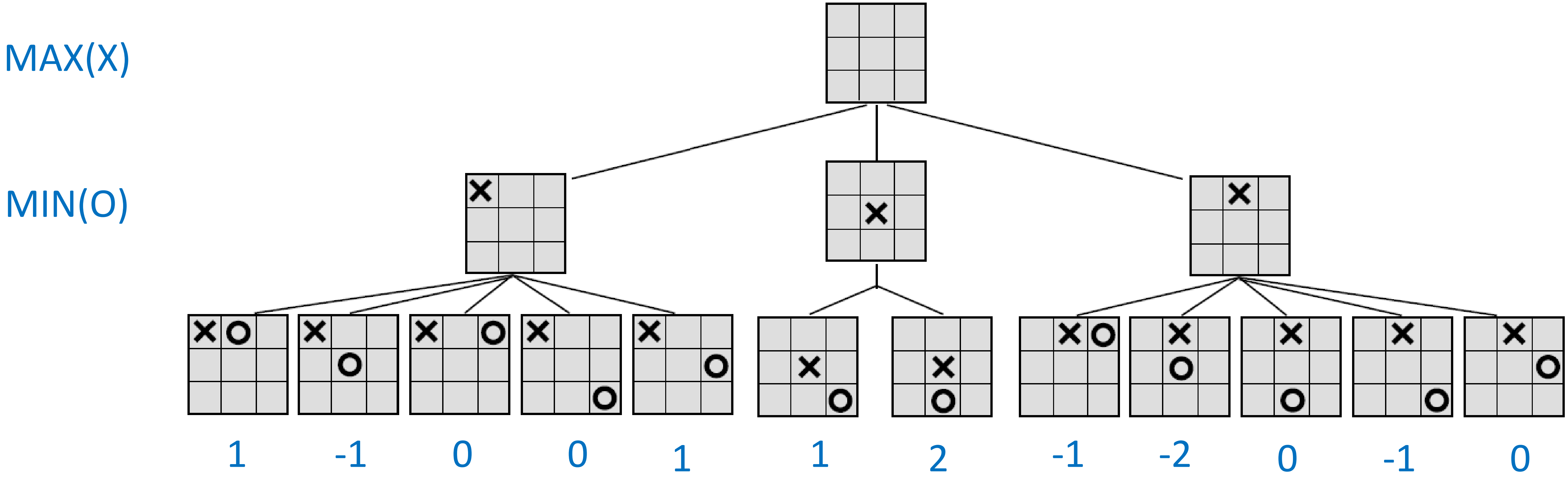
MIN(O)



Problem02

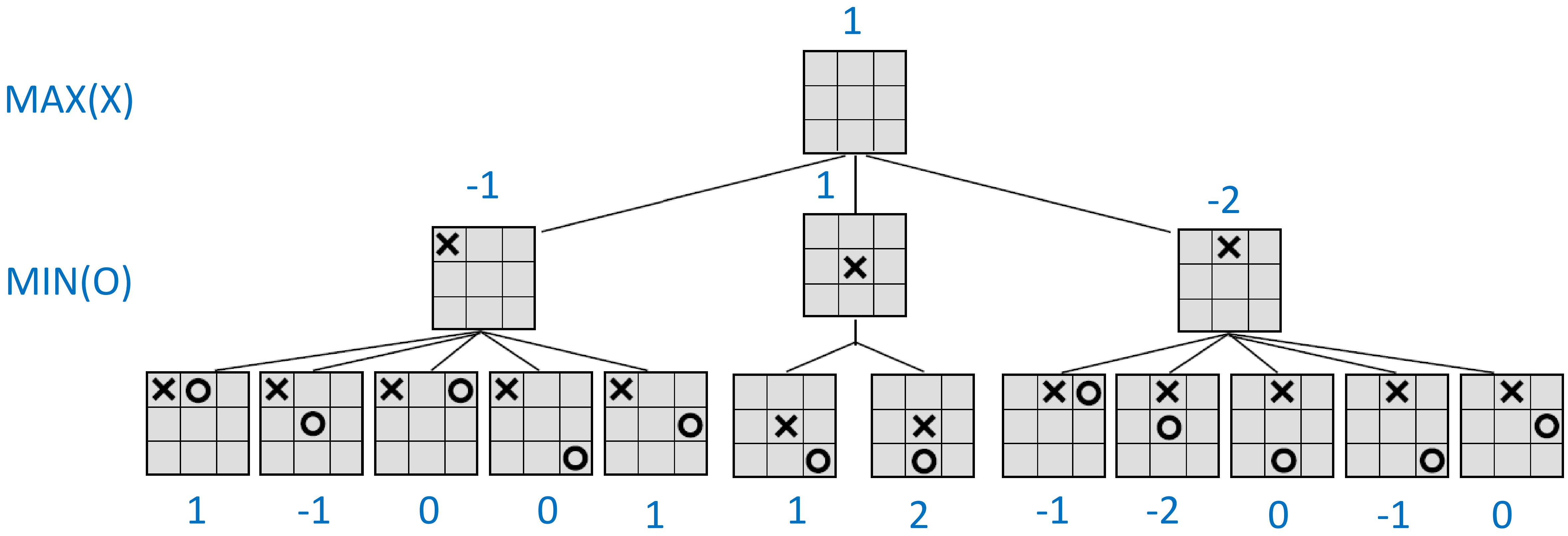
b) *Indicate on the game tree the estimated values of all positions at depth 2. Use the MiniMax algorithm to derive the backed-up values for positions at depths 1 and 0. Determine then the best starting move for the X player.*

With the heuristics of the value of each non-terminal position: $3X_2 + X_1 - 3O_2 - O_1$, the heuristics function becomes $X_1 - O_1$ as $X_2 = 0$ and $O_2 = 0$.



Problem02

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.*



NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Copyright Notice