SC3000 TUT

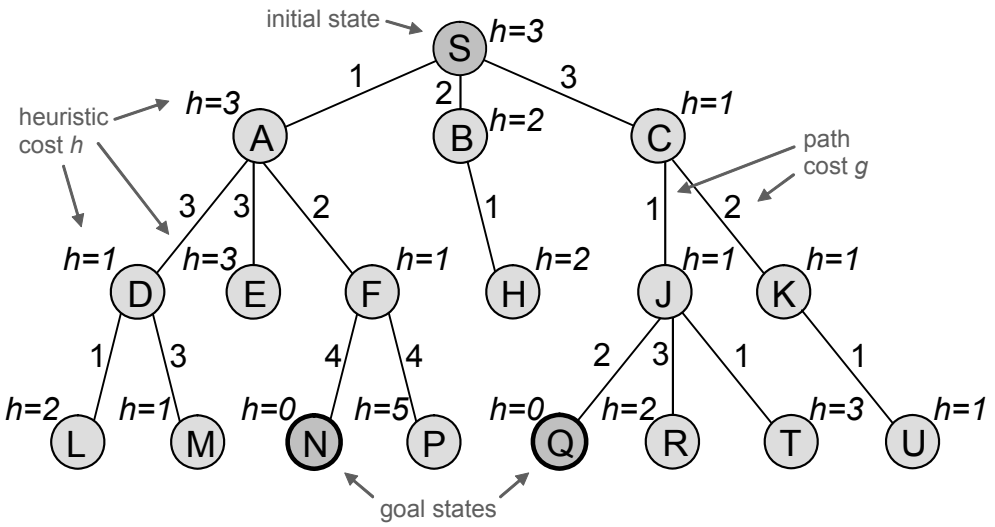
**1.1** Explain which *search algorithm* is most appropriate in the following situations:

(a) We have a very large search space with a large branching factor and with possibly infinite paths. We have no heuristic function. We want to find a path to the goal with minimum number of states.

(b) We have a state space with lots of cycles and links of varying costs. We have no heuristic function. We want to find the shortest path.

(c) Our search space is a tree of fixed depth and all the goals are at the bottom of the tree. We have a heuristic function and we want to find any goal as quickly as possible.

**1.2** Consider the search problem defined by the annotated search tree below.




(a) Apply the standard *A* search algorithm.* Draw all generated nodes, write their f-costs, and number expanded nodes in order of expansion. Assume that the children of a node are processed in alphabetical order, and that nodes of equal priority are extracted from the search queue in FIFO order.

(b) State how many nodes were generated and how many were expanded. Comment on the solution obtained and the *effectiveness* of the search. What do you think of the *heuristic function h* employed?

**1.3** The *w-A** search algorithm is a *weighted* variant of *A** that places more emphasis on the heuristic function by using the *f*-cost $f_W(n) = g(n) + w \times h(n)$, for any $w > 1$.

(a) Similarly to question 1.2a, apply the *w-A** search algorithm for $w = 2$.

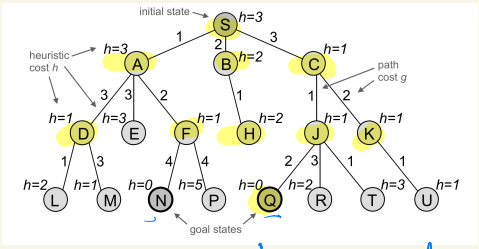
(b) Similarly to question 1.2b, comment on the *performance* and usefulness of the *w-A** search algorithm – in this case and in general.

TUT 3

1.1) a) DFS

b) Uniform-Cost Search

c) Greedy




1.2) a) S 0+3 = 3

A 1+3 = 4 B 4+2 = 4 C = 3+1 = 4

D = 5 E = 7 F = 7+1 = 4 H = 3+2 = 5 J = 4+1 = 5 K = 5+1 = 6

N = 7+0 = 7 Q = 6+0 = 6 U = 6+1 = 7

3+4+4+7 = 18 3+4+5+6 = 18

b) expanded: 10

generated: 18

1.3) a) S 0+2x3 = 6

A = 1+2x3 = 7 B = 2+2x2 = 6 C = 3+2x1 = 5

J = 4+2x1 = 6 K = 5+2x1 = 7

Q = 6+0 = 6

6+5+6+6 = 23

## *Most suitable search algorithm:*

a) *"very large search space"*: DFS, IDS, or A* - not BFS
*"large branching factor"* : DFS or IDS
*"possibly infinite paths"* : not DFS
*"no heuristic function"* : not A* et al
*"minimum no. of states"* : optimal, so BFS or IDS

→ Iterative Deepening Search (IDS) is best

b) *"lots of cycles"* : not DFS
*"varying costs"* : UCS or A* - not BFS, DFS
*"no heuristic function"* : not A* et al
*"shortest path"* : optimal, so UCS or A*

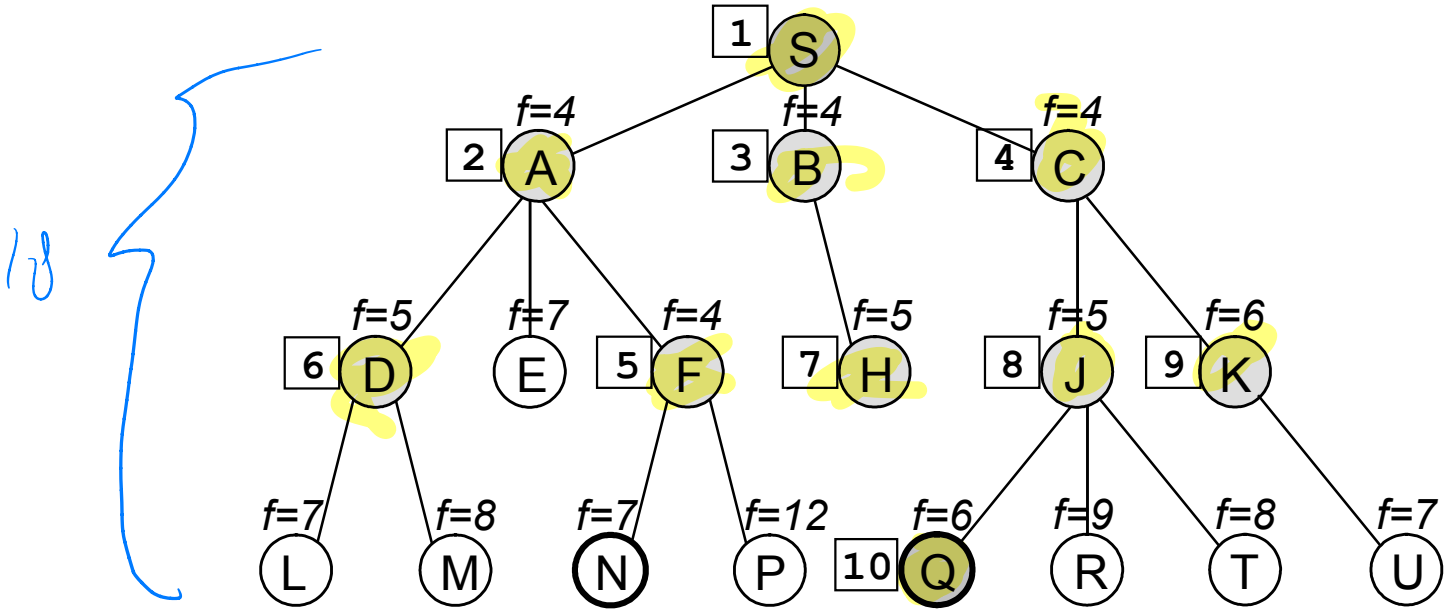
→ Uniform Cost Search (UCS / Dijsktra)

c) *"fixed depth tree"* : DFS, others
*"goals at the bottom"* : DFS - not BFS or IDS
*"heuristic function"* : Greedy Best First, not DFS
*"find any goal quickly"* : not opt., DFS or greedy Best First

→ Greedy Best First Search is best

note: local search algorithms such as Hill-climbing may fail to find a goal (no backtracking)

## A* search, solution and performance:



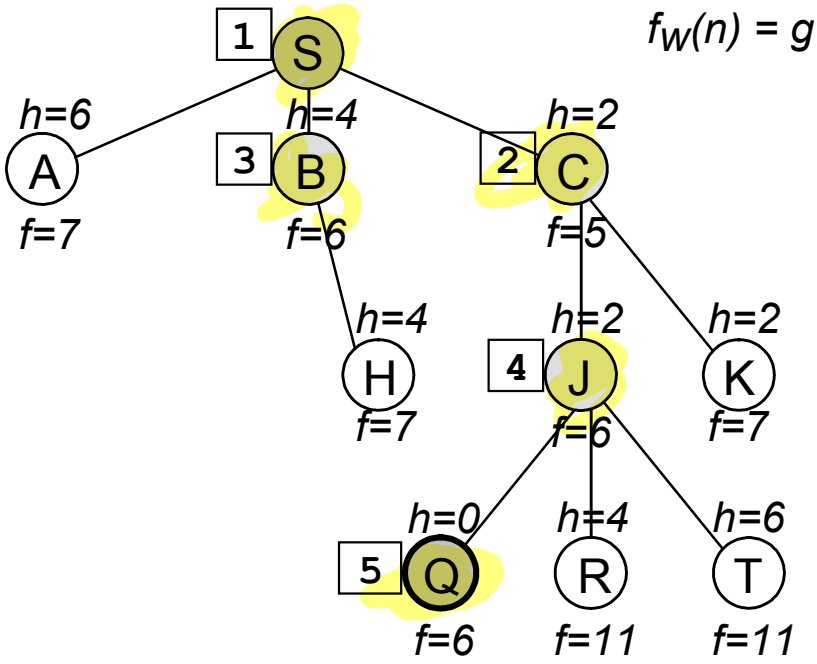

1. **S** (0+3=3)
2. **A** (1+3=4), **B** (2+2=4) , **C** (3+1=4)
3. **B**, **C**, **F** (3+1=4), **D** (4+1=5), **E** (4+3=7)
4. **C**, **F**, **D**, **H** (3+2=5), **E**
5. **F**, **D**, **H**, **J** (4+1=5), **K** (5+1=6), **E**
6. **D**, **H**, **J**, **K**, **E**, **N** (7+0=7), **P** (7+5=12)
7. **H**, **J**, **K**, **E**, **N**, **L** (5+2=7), **M** (7+1=8), **P**
8. **J**, **K**, **E**, **N**, **L**, **M**, **P**
9. **K**, **Q** (6+0=6), **E**, **N**, **L**, **M**, **T** (5+3=8), **R** (7+2=9), **P**
10. **Q**, **E**, **N**, **L**, **U** (6+1=7), **M**, **T**, **R**, **P**

queue

nodes generated: 18
nodes expanded: 10
optimal solution

*nearly exhaustive search* (!)
ill-guided → poor heuristics
(optimistic, misleading)

## Weighted A* search, solution and performance:




1. **S** (0+6=6)
2. **C** (3+2=5), **B** (2+4=6), **A** (1+6=7)
3. **B**, **J** (4+2=6), **A**, **K** (5+2=7)
4. **J**, **A**, **K**, **H** (3+4=7)
5. **Q** (6+0=6), **A**, **K**, **H**, **R** (7+4=11), **T** (5+6=11)

queue

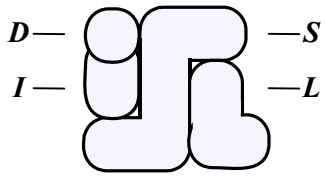
nodes generated: 10
nodes expanded: 5

half(!) *well-guided search* → much improved heuristics

*w*-A* – pros: faster, complete
cons: not optimal (no guarantee)

increase *w*? faster yet, less and less optimal (still better than greedy search!)

**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.

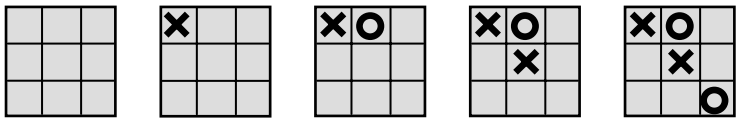



You are asked to use a *Depth-First Search* algorithm *with Forward Checking* to generate all possible color combinations for the logo. In both cases below, draw the complete search space, calculate the average branching factor and the total number of nodes, and comment on the efficiency of the approach.

(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.

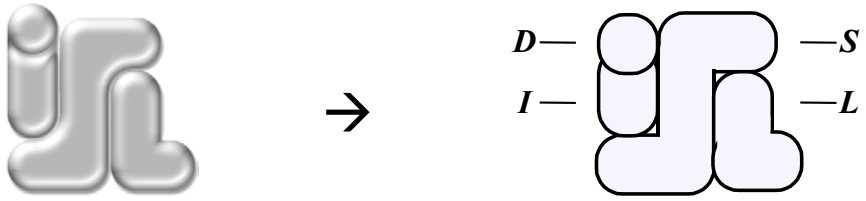


A *heuristics* for this game is proposed as follows. Let $X_n$ be the number of rows, columns, or diagonals with exactly $n$ X's and no O's. Similarly, let $O_n$ be the number rows, columns, or diagonals with $n$ O's and no X's. The heuristics gives any non-terminal position a value equal to: $3\ X_2 + X_1 - 3\ O_2 - O_1$. 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.

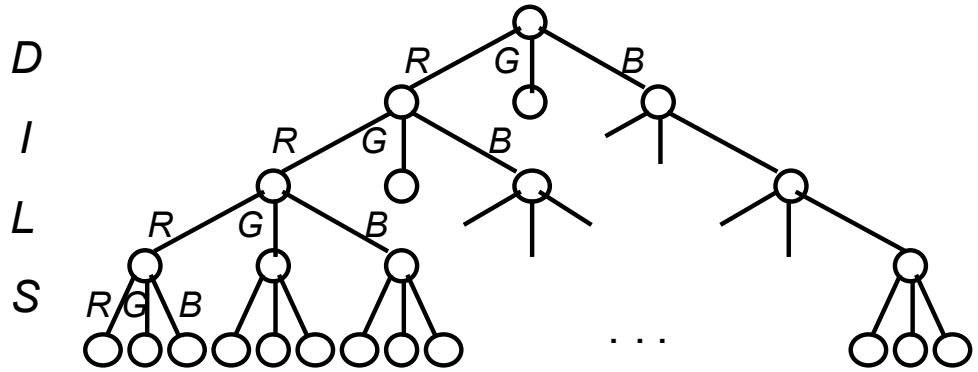
(b) Indicate on the game tree the *estimated values* of all positions at depth 2. Use the *MiniMax* algorithm to derive the backed-up values for positions at depths 1 and 0. Determine then the best starting move for the X player.

## ISL logo coloring problem:




## Depth-First Search (no forward checking):

search space:




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

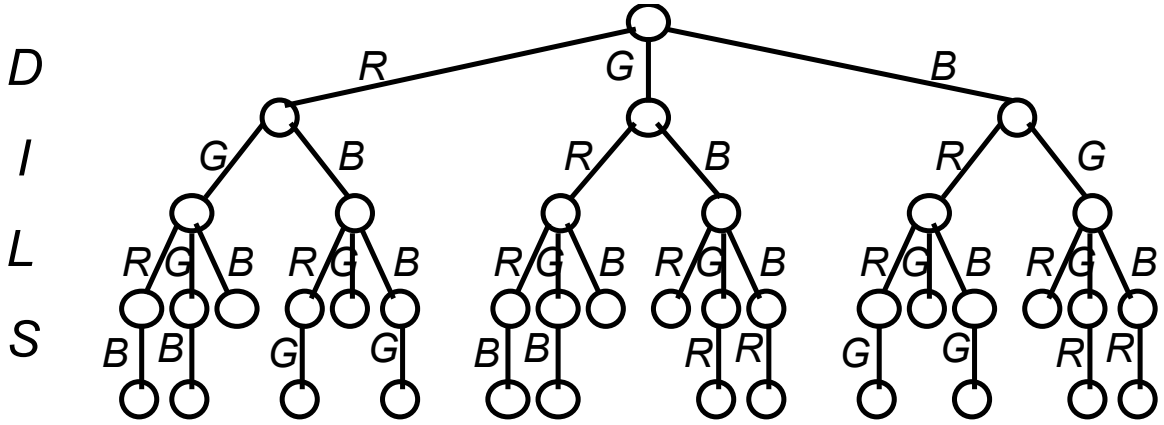
***no*** → need to generate all solutions (not just one), blind search generates invalid colorings, backtracks unnecessarily

## *DFS with forward checking, arbitrary ordering:*

logo colouring = Constraint Satisfaction Problem

forward checking: *take constraints into account to significantly prune the search space*

search space:




number of nodes: 1 + 3 + 3∗2 + 6∗3 + 6∗2 = 40

avg 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 colorings, little backtracking

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

## *General CSP heuristics, and their usefulness:*

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 nb of neighbors

→ useful to optimize the order of variable assigments

Least Constraining Value

select a value that leaves the largest choice of values for other constraint-related variables

i.e., color that less constrains other logo elements

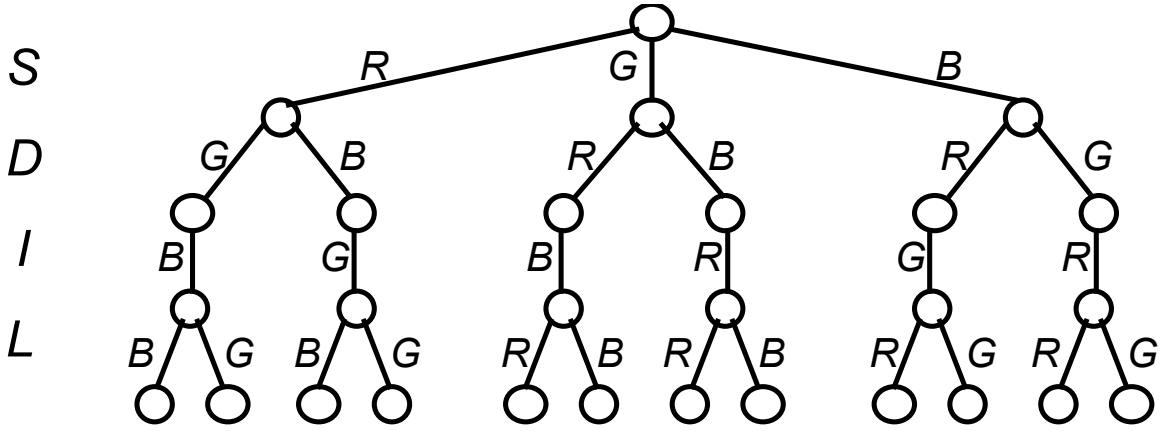
→ useful to prevent deadlocks, reduce backtracking

best heuristics: *Most Constraining Variable*

## *DFS with forward checking, heuristic ordering:*

most constraining logo element: *S* with 3 neighbors,
followed by *D* and *I* with 2 neighbors each,
then *L* with only 1 neighbor

search space:



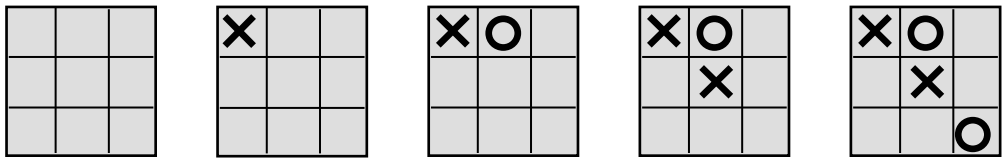

number of nodes: $1 + 3 + 3*2 + 6*1 + 6*2 = 28$

avg 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 colorings, no backtracking

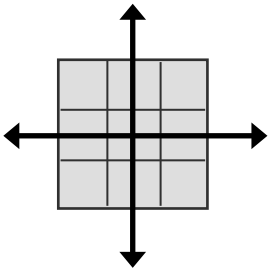
elements are assigned a color in *optimal* order

## *Depth-2 game tree for Tic-Tac-Toe:*

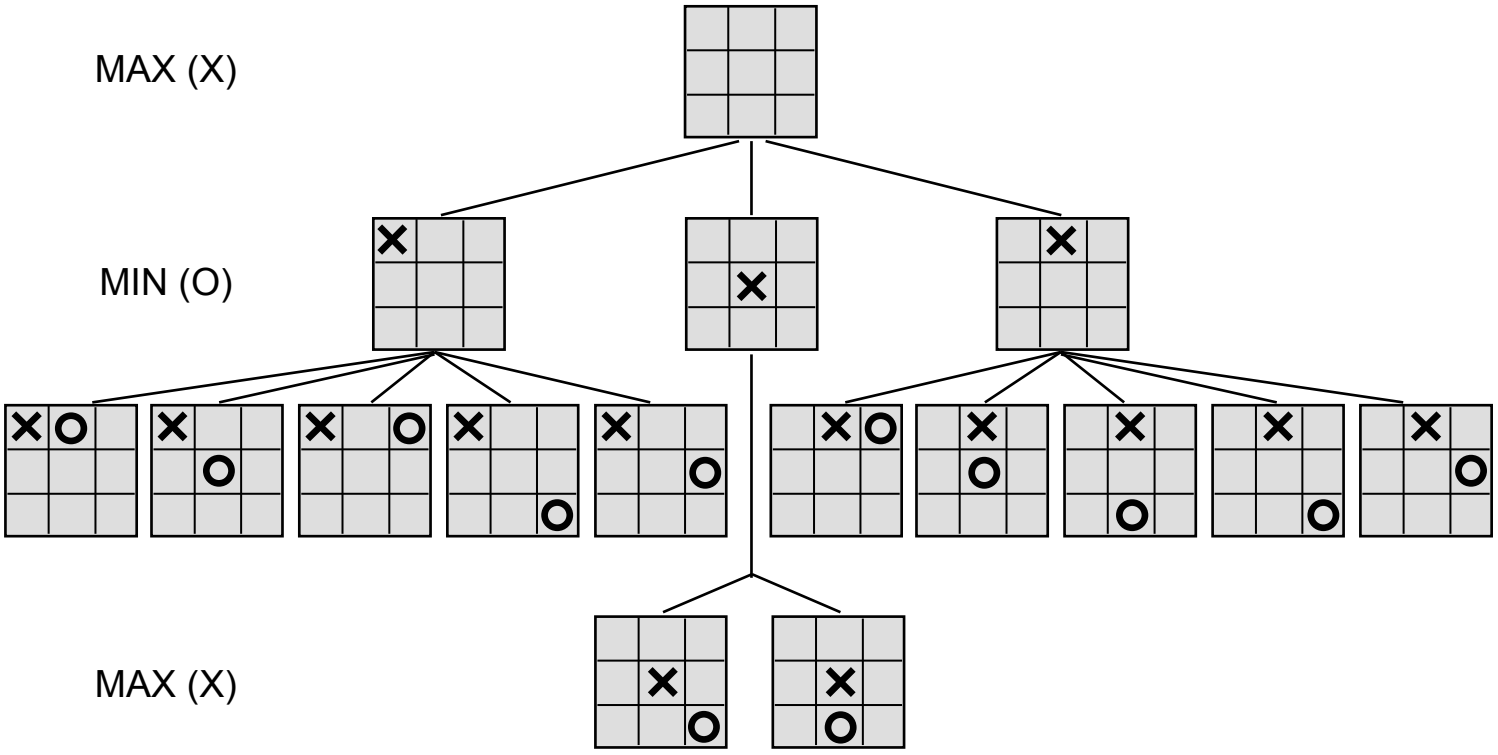


horizontal and vertical symmetry

thus only 3 moves at depth 1 (vs. 9)
and 12 moves at depth 2 (vs. 72)



search tree:

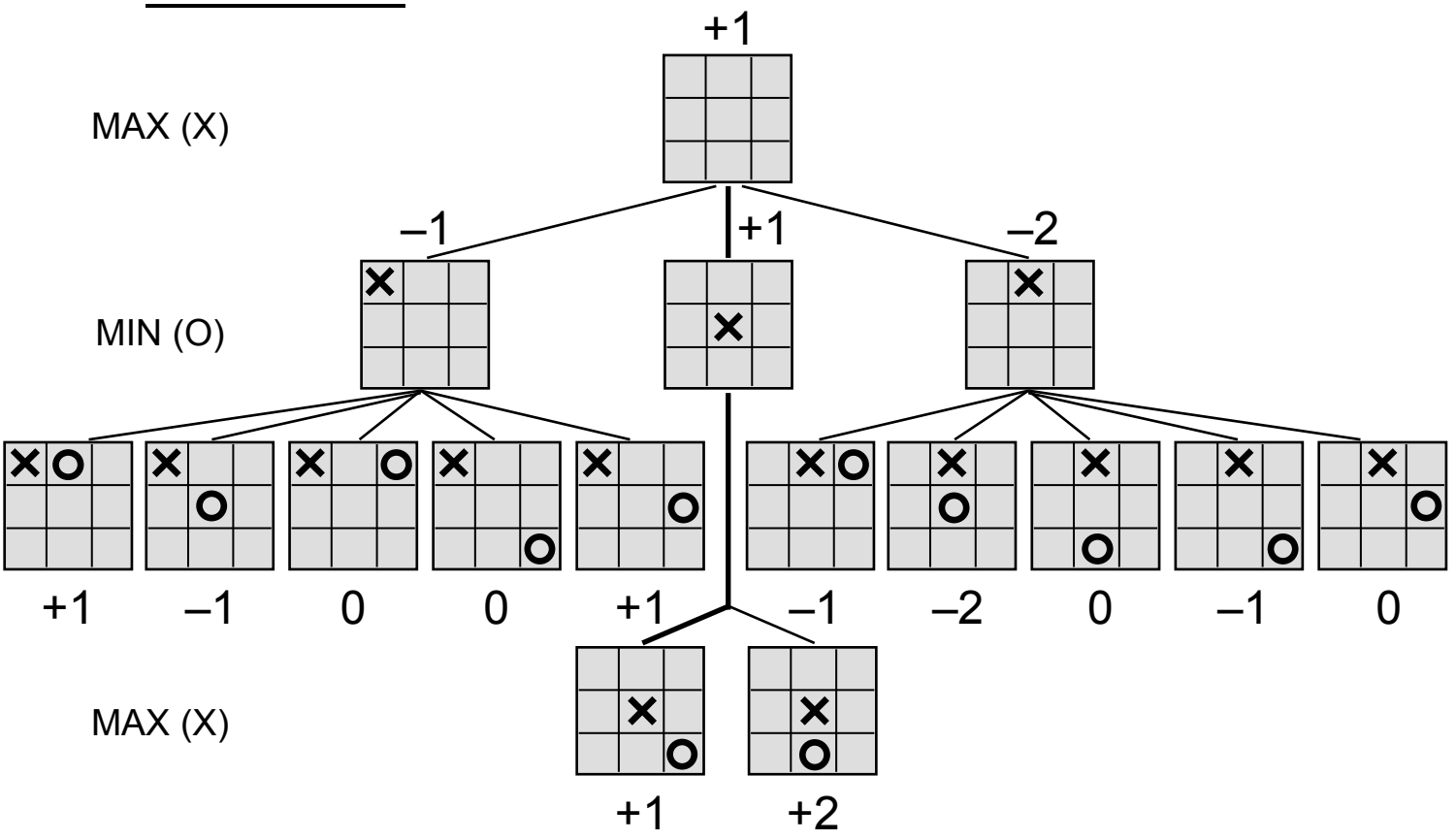


## *Heuristics for Tic-Tac-Toe:*

value of non–terminal nodes: $3\,X_2 + X_1 - 3\,O_2 - O_1$

at depth 2: $X_2 = O_2 = 0$ → $h = X_1 - O_1$

search tree:




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

fair game? no, whoever plays first has a strong advantage

**3.1** Assume that you are a manager of a warehouse (with a maximum capacity of $W$ items). Each month $t$, you know the current inventory (how many items left) in your warehouse. You might have a guess of the external demand in the next month ($t + 1$) with a distribution $p$ (the probability that the external demand are $j$ items is $p(D_t = j)$, $j = 0, 1, 2,\ldots$ ). Based on this information, you decide to order additional items from a supplier. The cost might come from the storing cost of items in warehouse. Your objective is to maximize the profit. Use your own parameters for fixed costs to buy and store for each item and a fixed selling price.

Please write an MDP formulation for the above problem.
Hint: Decision epochs are made at the beginning of each month, hence all events (more items arrive, fill external orders) would make states change. Actions are the amount of an order.

**3.2** This Gridworld MDP operates like to the one we saw in class. The states are grid squares, identified by their row and column number (row first). The agent always starts in state (1,1), marked with the letter S. There are two terminal goal states, (2,3) with reward +5 and (1,3) with reward -5. Rewards are 0 in non-terminal states. (The reward for a state is received as the agent moves into the state.) The transition function is such that the intended agent movement (North, South, West, or East) happens with probability .8. With probability .1 each, the agent ends up in one of the states perpendicular to the intended direction. If a collision with a wall happens, the agent stays in the same state.

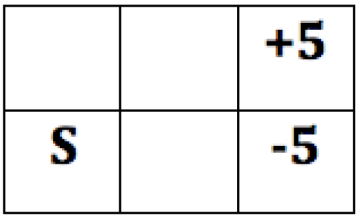


Figure (a) Gridworld MDP.

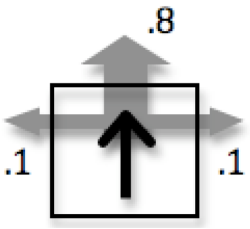


Figure (b) Transition function.

(a) Suppose the agent knows the transition probabilities. Give the first two rounds of value iteration updates for each state, with a discount of 0.9. (Assume $V_0$ is 0 everywhere and compute $V_i$ for times $i = 1, 2$). (Assume values of termination states ((1, 3) and (2, 3)) are always 0)

(b) Suppose the agent does not know the transition probabilities. What does it need to be able do (or have available) in order to learn the optimal policy?

(c) The agent starts with the policy that always chooses to go right, and executes the following three trials: 1) (1,1) – (1,2) – (1,3), 2) (1,1) – (1,2) – (2,2) – (2,3), and 3) (1,1) – (2,1) – (2,2) – (2,3). What are the Monte Carlo estimates for states (1,1) and (2,2), given these traces (assuming that the discount factor is 1)?

(d) Using a learning rate of .1 and assuming initial values of 0, what updates does the Q-learning agent make after trials 1 and 2, above?

3.1 TNT MDP

capacity W

month t

- {0, 1, ... W} action a order additional?

{0, 1, ... W} State S Current inventory / item left in WH

external demand J

i = current iter

Policy

Reward R(S_t, a_t)

↳ Cost of buying at Buy(a_t)

↳ Cost for storing (S_t + a_t) / Store(S_t + a_t)

↳ Selling price D_t f(D_t)

Sell(S_t + a_t) = $\sum_{d=0}^{S_t+a_t} P(D_t = d)\, f(d)$

P(s'|s,a) Transition model assume (t+1) P(D_t = j)

ses:

- If $j > i + a$, then $T(j|s = i, a) = 0$. That means even after sale the remaining in the warehouse cannot exceed the current capacity.
- If $j \leq i + a$ and $j > 0$, that means the demands at time $t$ does not exceed the capacity. Hence $T(j|i, a) = p(D_t = i + a - j)$
- If $j = 0$, that means the demand is equal to or exceeds the capacity. Hence

$$T(j|i, a) = p(D_t \geq i + a) = \sum_{d=i+a}^{\infty} p(D_t = d)$$

3.2 action (North, South, west, East)

State ((1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3))

Reward 0, +5, -5 R(s, a, s')

Transition

γ = 0.9

$V_{i+1}(s) = \max_{a} \sum_{s' \in S} P(s'|s,a)\left[R(s,a,s') + \gamma V_i(s')\right]$ (0.8 → 0.1, 0.1 ← 0.1)

$V_0 = 0$

| | (1,1) | (1,2) | (1,3) | (2,1) | (2,2) | (2,3) |
|---|---|---|---|---|---|---|
| $V_0$ | 0 | 0 | 0 | 0 | 0 | 0 |
| $V_1$ | 0 | 0 | 0 | 0 | 4 | 0 |
| $V_2$ | | | 0 | | | 0 |

Q((1,1), N) = stay + right + left + down

= 0.8(0) + 0.1(stay 0 + 0.5 + 0) + 0.1(0 + 0 + 0)

Q((1,2), S) = 0 + 0.1(0.5 + 0 + 0) + 0.1(0) + 0.8(0)

Q((1,2), W) = 0.1(0) + 0 + 0.8(0) + 0.1(0)

Q((1,2), E) = 0.1(0) + 0.8(stay 0) + 0 + 0.1(0)

9

Q((1,1), N) = stay + up + right

PC((1,1),(1,1)|N) · (1 − 0.1 − 0.8) × [0 + 0] = 0

PC((1,2),(1,1)|N) · 0.8 × 0 = 0

PC((1,1),(2,1)|N) · 0.1 × 0 = 0

Q((1,1), E) = stay + up + right

down = 0.1(0) + 0.1(0) + 0.8(0) = 0

Q((1,1), W) = stay + up + right

down = 0.1(0) + 0.1(0) + 0(0) = 0

Q((1,1), S) = stay + up + right

down = 0.1(0) + 0(0) + 0.1(0) = 0

$V_2$

$Q_2((1,2), N)$ = stay + left + right + up
$= 0 + 0.1(0) + 0.1(-5+0) + 0.8(0) = -0.5$

$P((1,2) | (2,2) | N) \left[ R((2,2), N, (1,2)) + \gamma V(1,2) \right]$

$Q_2((1,2), S)$ = down + up + left + right
$0.8(0) + 0 + 0.1(0) + 0.1(-5)$

$Q_2((1,2), E)$ = down + up + left + right
$0.1( \ ) + 0.1( \ ) + 0 + 0.8(-5+0.7)$
$-1.12$

$Q_2((1,2), N)$ = up + down + left + right
$= 0.8(0 + 0.9(4)) + 0 + 0.1(-5) + 0.1(0)$
$= 2.38$

$Q_2((2,2], N)$ = stay + left + right + down
$= 0.8(0 + 0.6) + 0.1(0.6) + 0.1(5 + 0.6) + 0$
$=$

20] $(1,1) \rightarrow (1,2) \rightarrow (1,3)$

$(1,1) \rightarrow (1,2) \rightarrow (2,2) \rightarrow (2,3)$

$(1,1) \rightarrow (2,1) \rightarrow (2,2) \rightarrow (2,3)$

Monte carlo $(1,1)$
$(2,2)$

# MDP and RL Tutorial – Solutions

**3.1** One form of MDP formulation $\{S, A, T, R\}$ is as follows.

State space is $S = \{0,1,2, \dots, W\}$

Action space $A = \{0,1,2, \dots, W\}$

The reward term $R(s_t, a_t)$ consists of three components:

- the cost of buying $a_t$ items are $Buy(a_t)$
- cost for storing $(s_t + a_t)$. This cost is fixed and presumably it is equal to $Store(s_t + a_t)$.
- Assume the selling price of $D_t$ items is $f(D_t)$. The total sale price is

$$Sell(s_t + a_t) = \sum_{d=0}^{s_a + a_t} p(D_t = d) f(d)$$

In summary, the reward function is

$$R(s_t, a_t) = Sell(s_t + a_t) - buy(a_t) - Store(s_t + a_t)$$

The transition function $T(s' = j | s = i, a)$ has three cases:

- If $j > i + a$, then $T(j | s = i, a) = 0$. That means even after sale the remaining in the warehouse cannot exceed the current capacity.
- If $j \leq i + a$ and $j > 0$, that means the demands at time $t$ does not exceed the capacity. Hence $T(j | i, a) = p(D_t = i + a - j)$
- If $j = 0$, that means the demand is equal to or exceeds the capacity. Hence

$$T(j | i, a) = p(D_t \geq i + a) = \sum_{d=i+a}^{\infty} p(D_t = d)$$

**3.2** (a) Apply the Bellman backups $V_{i+1}(s) = \max_a(\sum_{s'} T(s,a,s')(R(s,a,s') + \gamma V_i(s'))$ twice. We just show the computation for the max actions. Most of the terms will be zero, which are omitted here for compactness.

| $S =$ | (1,1) | (1,2) | (1,3) | (2,1) | (2,2) | (2,3) |
|---|---|---|---|---|---|---|
| $V_0(S) =$ | 0 | 0 | 0 | 0 | 0 | 0 |
| $V_1(S) =$ | 0 | 0 | 0 | 0 | $0.8 \times 5 = 4.0$ | 0 |
| $V_2(S) =$ | 0 | $0.9 \times 0.8 \times 4 + 0.1 \times -5 = 2.38$ | 0 | $0.8 \times 0.9 \times 4.0 = 2.88$ | $0.8 \times 5 = 4.0$ | 0 |

(b) The agent must be able to explore the world by taking actions and observing the effects.

(c) To compute the estimates, average the rewards received in the trajectories that went through the indicates states.

$$V((1,1)) = ((-5 + 5 + 5))/3 = 5/3 = 1.666$$
$$V((2,2)) = ((5 + 5))/2 = 5$$

(d) The general Q-learning update is:

$$Q_{new}(s,a) = Q_{old}(s,a) + \alpha[r + \gamma \max_{a'} Q_{old}(s',a') - Q_{old}(s,a)]$$

After trial 1, all of the updates will be zero, expect for:

$$Q\big((1,2), right\big) = 0 + .1\,(-5 + 0.9 \times 0 - 0) = -0.5$$

After trial 2, the non-zero updates will be:

$$Q\big((1,2), right\big) = -0.5 + .1(0 + 0.9 \times 0 - (-0.5)) = -0.45$$
$$Q\big((2,2), right\big) = 0 + .1(5 + 0.9 \times 0 - 0) = 0.5$$