

מבוא לבינה מלאכותית - אביב - 236501

תרגיל בית 2

תאריך הגשה : 25.06.2023

<u>שם פרטי</u>	<u>שם משפחה</u>	<u>תעודת זהות</u>
<u>נוואף</u>	<u>סלמאן</u>	<u>207608985</u>
<u>עומר</u>	<u>שרפי</u>	<u>315271239</u>

nawafsalman@campus.technion.ac.il

Omar.sharafy@campus.technion.ac.il

Part A

1)

- State Space (S): The state space consists of all possible configurations of the game. In this game, it includes the positions of the robots, packages, destinations, and charging stations on the 5x5 grid board, as well as the battery, remaining moves for each robot and credit units of each robot.

- i. RobotLocation0, RobotLocation1 $\in [0,24]$.
- ii. RobotFuel0, RobotFuel1 $\in \mathbb{N} \cup \{0\}$.
- iii. RobotCredits0, RobotCredits1 $\in \mathbb{N} \cup \{0\}$.
- iv. RobotPack0, RobotPack1 $\in \{\text{pack0}, \text{pack1}, \text{none}\}$ (Which pack is already picked up by the robot).
- v. Pack0, Pack1 $\in [0,24]$.
- vi. Dest0, Dest1 $\in [0,24]$.
- vii. RemainingMoves $\in \mathbb{N} \cup \{0\}$.

- Initial State (I): The initial state represents the starting configuration of the game, where the robots start with a certain amount of battery and credit units, and the packages, destinations, and charging stations are placed on the grid.

- Operators (O): The actions available to the robots include "move north," "move south," "move east," "move west," "pick up," "drop off," and "charge." These actions allow the robots to move in the grid, pick up and drop off packages, and charge their batteries at the charging stations.

- Goal State (G): The goal state is reached when one of the robots runs out of battery or when the maximum number of steps for each robot is reached. The goal is to score more points than the opponent robot by delivering packages to their destinations.



2)

We need a heuristic that will give importance to Manhattan distance from the package. if robot already has package, we prioritize delivering current package to its destination. Also, the heuristic will give importance to picking up a package if robot can. And will give higher importance to dropping off packages if arrived to destination (higher, because otherwise the robot will carry the package forever).

Define:

1- $\text{max_remaining_battery_to_package} = \text{current_battery} - \text{Manhattan_distance_to_nearest_package}$

2- $\text{max_remaining_battery_to_dest} = \text{current_battery} - \text{Manhattan_distance_to_current_package_dest}$

3- $\text{pickup_reward}(s) = \begin{cases} 5, & \text{if in state } s, \text{ robot is carrying a package} \\ 0, & \text{otherwise} \end{cases}$

4- $\text{dropoff_reward}(s) = 6 * \text{current_robot_credit}$

5- $\text{remaining_battery_to_dest}(s) = \begin{cases} \text{max_remaining_battery_to_package}(s), & \text{if in state } s, \text{ robot is not carrying a package} \\ \text{max_remaining_battery_to_dest}(s), & \text{otherwise} \end{cases}$


Final_heuristic:

$$h(s) = \text{remaining_battery_to_dest}(s) + \text{pickup_reward}(s) + \text{dropoff_reward}(s)$$

according to this heuristic, if a robot can pick up or drop off a package in state s, it will. otherwise, it will prioritize going to the nearest package to pick.

The numbers 5 and 6 were chosen according to some experiments. (running greedy with different values).

4)


The main disadvantage of the Greedy algorithm, compared to Minimax, is that it does not consider the long-term consequences of its actions. It focuses solely on the immediate gain  benefit without considering the potential future states or outcomes. This can lead to suboptimal decisions in complex game scenarios where a short-term gain might lead to a disadvantageous position in the future. In contrast, Minimax considers the entire game tree and explores different possibilities to make decisions that optimize long-term outcomes. Also, **greedy does not consider the other player's decisions** (using our heuristic).

PART B

1)

Advantages of an Easy-to-Calculate Heuristic:

- **Computational Efficiency**: Easy-to-calculate heuristics can be computed quickly, allowing for faster decision-making in real-time or resource-limited scenarios. This is especially important in situations where time is a critical factor, and there is a need to make decisions promptly.

- **Lower Resource Requirement**  Easy-to-calculate heuristics typically require fewer computational resources, such as memory or processing power, compared to complex heuristics. This makes them more suitable for environments with limited resources, where minimizing resource usage is essential and also allows us to instigate the tree deeper.


Disadvantages of an Easy-to-Calculate Heuristic:

- **Lack of Precision**: Easy-to-calculate heuristics often provide less accurate estimations or predictions of the true value or quality of a state. They may overlook important factors or nuances of the problem, leading to suboptimal decisions. This can be particularly problematic in complex or strategic scenarios where a high level of accuracy is required.


- **Limited Information**: Easy-to-calculate heuristics might consider only a subset of relevant information or features in the problem domain. They may not capture the full complexity or richness of the state space, resulting in less informed decisions. This limitation can lead to missed opportunities or suboptimal strategies.

2)

This doesn't necessarily mean that she has a bug.

minimax works with the  assumption of perfect opponent, this means it does not guarantee finding the optimal move in all situations, but the optimal move against perfect opponent whom only want to make us lose.

4)

- a. If every agent wants to win the game and doesn't care about us, then every node in the minimax tree should be a max node. Because every agent will choose the move that will maximize its utility.
- b. if all agents want me to lose the game, then every node in minimax tree should be min node except our node will be max. because every agent will choose the move that will minimize our utility.
- c. if each agent wants the agent next in line to win, then every node in minimax tree should be max node with some change: it will take the maximum value of the children of the nodes in the next line. Because each agent will choose the move that will maximize the utility of the agent next line. 

PART C

2)

Yes, the agent will behave differently.

if given very little resources, the regular RB minimax agent won't be a lot different from the improved greedy, because it won't have a lot of time to decide for next move and it will depend mainly on the heuristic.

but we expect the alpha-beta algorithm to be faster and find the optimal next move even when it doesn't have many resources.

PART D

1)

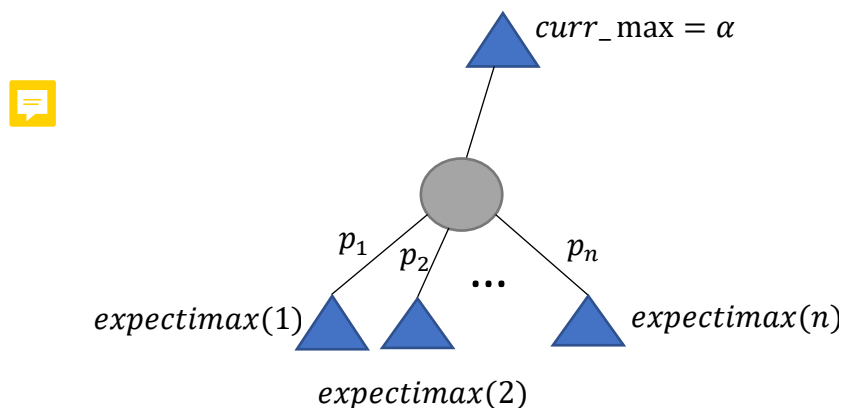
When using the Expectimax algorithm against an agent that plays completely randomly, we should use equal probability for every possible action.

By treating the opponent's actions as equally likely, the Expectimax algorithm can effectively explore the game tree and evaluate the expected value of each possible move. This allows it to make informed decisions based on the average outcome of the game rather than assuming a specific opponent strategy.

2)

We can prune max nodes as we learned in class (alpha-beta pruning).

for probabilistic nodes, we can know the upper limit that node can have without iterating over all its children and calculating the accurate expected value:



After iterating on i children of the probabilistic node, we now that the upper limit of the expected value is : $current_e + (1 - p_sum)$

$current_e$: the expected value of the children that we already iterated over them.

p_sum : the sum of the probabilities : $(p_1 + p_2 + \dots + p_i)$

The upper limit on the expected value is $current_e + (1 - p_sum)$ because we will get this value only if the remaining nodes that we didn't iterate over them yet all have a maximum expectimax value = 1 (max heuristic value is 1):

$$\begin{aligned}
expected_value &= current_e + \sum_{k=i}^n p_i * expectimax(i) \leq current_e + \sum_{k=i}^n p_i * 1 \\
&= current_e + 1 - \sum_{k=1}^i p_i = current_e + (1 - p_sum) \\
&= upper_limit
\end{aligned}$$

Algorithm:

when reaching $upper_limit < \alpha$, the value is irrelevant to parent max node so we can return (prune remaining children):

Init : $i = 1, \quad current_e = 0, \quad p_sum = 0$

For $i : 1 \rightarrow n$:

$current_e += p_i * expectimax(i)$
 $p_sum += p_i$
 $upper_limit = current_e + (1 - p_sum)$
if $(upper_limit \leq \alpha) : break$

return $current_e$



PART E

(1

א. מקדם הסיעוף לא יגדל, יתכן אפילו שיקטן, זאת כיוון שלא הגדלנו מכל מצב את מספר המצבים שאפשר להגיע אליהן, לא הוספנו פעילות של הגעה למצבים חדשים וכו'. ולכן מכל מצב נפתח את אותם מצבים שהיו לפני (אולי אפילו פחות, כיוון שלא נוכל לפתח צמתים שהם "לנוע בכיוון מחסומים") – ובעצם הוא נשאר אותו דבר – מכל נמצב לא שינינו את קבוצת המצבים שניתן ממנו להגיע אליהם (אולי חסמנו בנים ולא נוכל לפתח אותם – אבל הם קיימים – בדומה לholes משב. 1).

ב. מקדם הסיעוף יגדל, נסביר, כיוון שהגדלנו מכל מצב את מספר המצבים שאפשר להגיע אליהן, דרך הוספת פעילות ה"הנחת מחסום" אשר תיתן לכל רובוט עוד אפשריות מהמצב שבו הוא נמצא (5*5 פעולות חדשות מינוס המיקום שלו ושל הרובוט השני, מינוס מיקומי החבילות ותחנות הטעינה – בהנחה שלא ניתן להניח שם דברים – אז הוספו בהתחלה כ-8 – 25 מצבים חדשים להגיע אליהן מהמצב שלנו –), ולכן יש לנו יותר מצבים ל"הגיע" אליהן, ולכן הגדלנו את מקדם הסיעוף.



נשים לב שבמקרה המיוחד שהרובוטים עומדים על החבילות / על התחנות וכו' נקבל 4 – 25 ולכן מקדם הסיעוף המקסימלי: (אם לכל רובוט יש את החבילה שלו והוא בעמדת טעינה)

$$b' = 28$$

$b' = 7 + N$ where 'N' represents the number of empty slots on the board.

]

2)

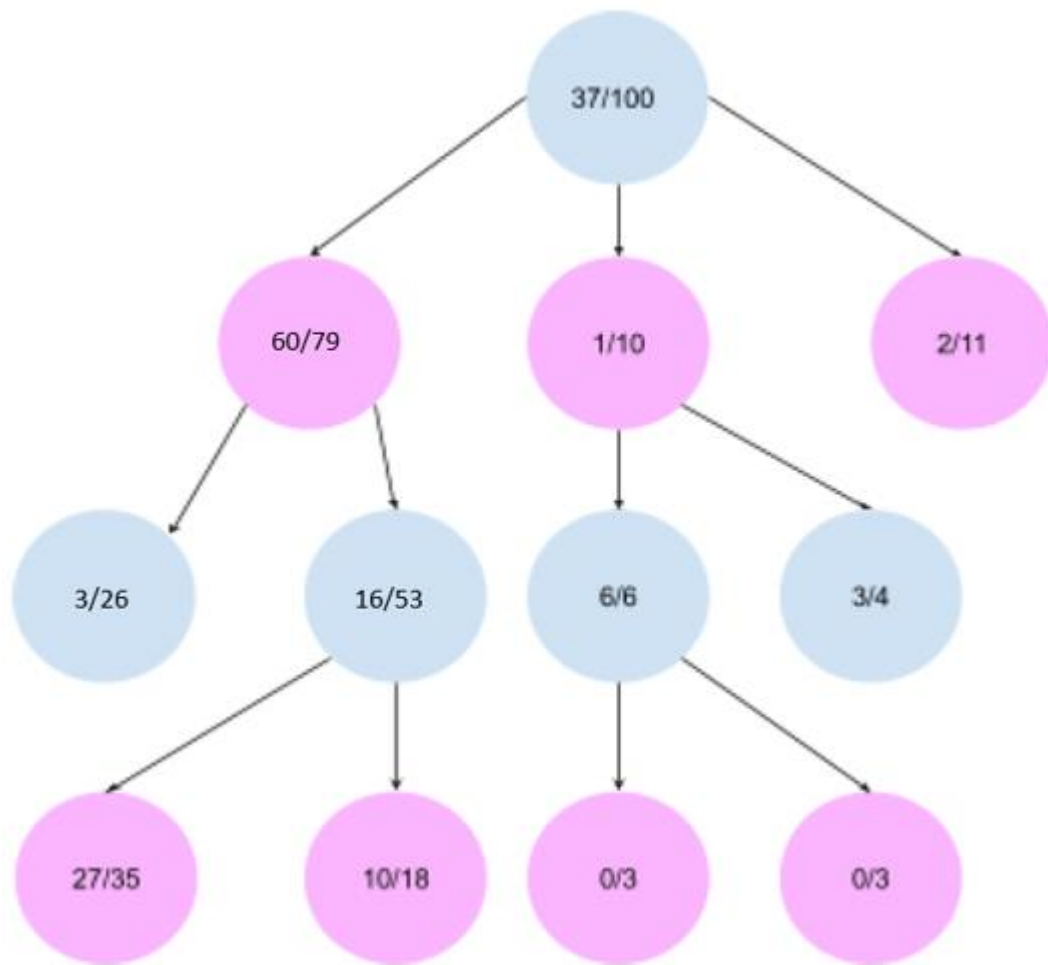
a. No, the algorithms in the previous sections (minimax, expect_max, alpha-beta) all have exponential runtime complexity in terms of the branching factor, so none of them will provide a reasonable runtime with such a large branching factor.



b. To maintain a reasonable running time, we would use Monte Carlo Tree Search (MCTS), which can handle larger branching factors more efficiently

PART F

1)



2)

► First action

- **Left node** $\frac{60}{79} + \sqrt{2} \times \sqrt{\frac{\ln(100)}{79}} = 1.1$
- **Middle node** $\frac{1}{10} + \sqrt{2} \times \sqrt{\frac{\ln(100)}{10}} = 1.06$
- **Right node** $\frac{2}{11} + \sqrt{2} \times \sqrt{\frac{\ln(100)}{11}} = 1.09$

► Second action

- **Left node** $\frac{3}{26} + \sqrt{2} \times \sqrt{\frac{\ln(79)}{26}} = 0.69$
- **Right node** $\frac{16}{53} + \sqrt{2} \times \sqrt{\frac{\ln(79)}{53}} = 0.71$

► Third action

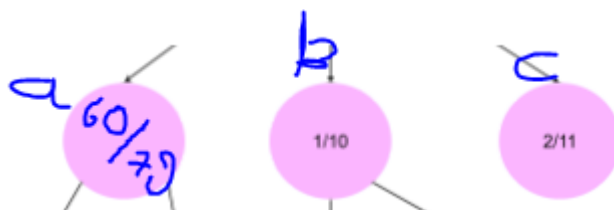
- **Left node** $\frac{27}{35} + \sqrt{2} \times \sqrt{\frac{\ln(53)}{35}} = 1.24$
- **Right node** $\frac{10}{18} + \sqrt{2} \times \sqrt{\frac{\ln(53)}{18}} = 1.21$

Then the left node is going to be selected since it is the biggest.



3)

assuming these are a , b and c



Tal winning means both the numerator and the denominator would increase by 1 for the node we choose , and the number of games for all nodes

Lets look at a and c as we would choose a but c is the closesr to it value

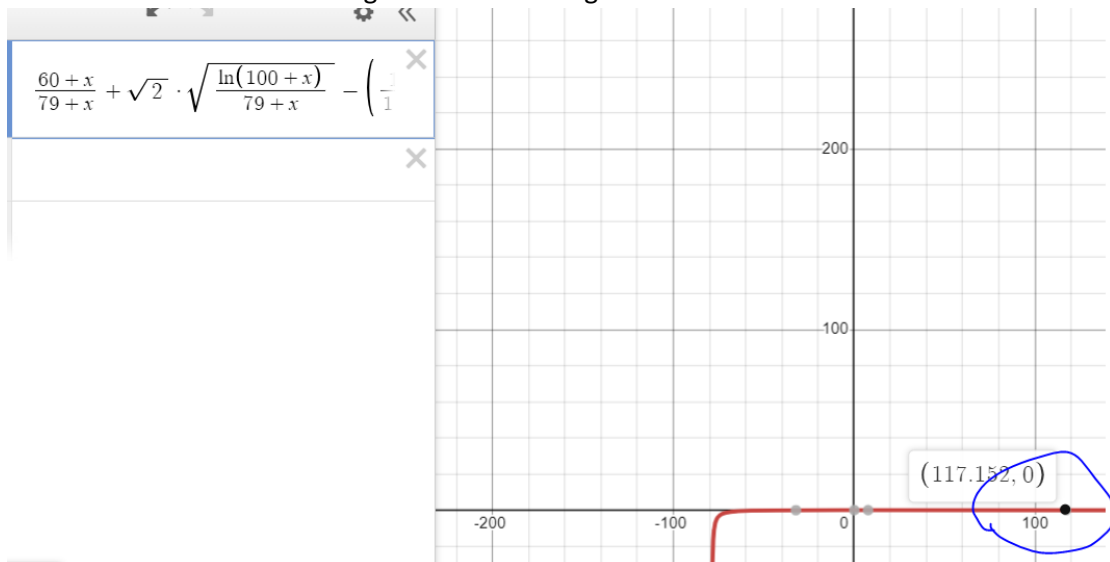
$$\frac{60+x}{79+x} + \sqrt{2} * \sqrt{\frac{\ln(x+100)}{79+x}} \leq \frac{2}{11} + \sqrt{2} * \sqrt{\frac{\ln(x+100)}{11}}$$

When this happened , we will move from a to c



So we will play 37 times to move from a to c

But about from a to b ? we might move before right ? if we check



So it will happen much later , so the answer is $x = 37$

- But if we choose another node to start with and win (for example b) , we notice that : Left node without additional wins: $\frac{60}{79} + \sqrt{\frac{2 \ln(101)}{79}} = 1.1013$
- Middle node with 1 additional win: $\frac{2}{10} + \sqrt{\frac{2 \ln(101)}{11}} = 1.116$
- Right node: $\frac{2}{12} + \sqrt{\frac{2 \ln(101)}{12}} = 1.04$

And so, we can see that with even 1 added victory the ancestor already changes there for $x=1$.

4)

we want to favor exploration more than exploitation compared to the previous formula

$$\frac{\text{exploration}}{\text{exploitaion}} = \frac{c * \sqrt{\frac{\ln(N(s.\text{parent}))}{N(s)}}}{\frac{U(s)}{N(s)}} = \frac{c}{U(s)} \sqrt{\ln(N(s.\text{parent})) * N(s)}$$

the formula uses $N(s)$ which you have access to and can change , we see from the math that if we for example make $N(s) *= N(s) * 4$ then we will favor exploration over exploitation by 2 compared to the previous formula .

so would like to increase $N(s)$.

