תרגיל בית 2 מבוא לבינה מלאכותית

חיפוש רב סוכנים

בינה מחסנים

מגישים: 31517334 - 206458390

חלק א - **ImprovedGreedy**

1. **(יבש: 4 נק') כפי שלמדתם, אנו מגדירים בעייה במרחב בתור רבעייה (S, O, I, G). הגדירו פורמלית (הסבירו כיצד נראים ערכי הרבעייה) את המשחק המתואר לכם ע"פ הנתונים שאתם מקבלים מהסביבה.**

S = {(a0_loc, a0_battery, a0_credit, a0_package, a1_loc, a1_battery, a1_credit, a1_package, p0, p1, d0, d1, x0, x1, c0, c1, t) |
a0_loc, a1_loc, p0, p1, d0, d1, x0, x1, c0, c1 ∈{0,…,4}x{0,…,4},
a0_battery, a0_credit, a1_battery, a1_credit, t ∈ $\mathbb{N}^+$,
a0_package, a1_package ∈ $\{True, False\}$,
a0_loc ≠a1_loc, c0 ≠c1}

S היא קבוצת הטאפלים:
(a0_loc, a0_battery, a0_credit, a0_package, a1_loc, a1_battery, a1_credit, a1_package, p0, p1, d0, d1, x0, x1, c0, c1, t)
תחת ההגבלה ששני הסוכנים לא באותו התא ושתי תחנות ההטענה לא באותו התא.

O = {right, left, up, down, charge, drop, pickup}
O קבוצת הפעולות האפשריות (המגבלות על חוקיות האופרטורים נמצאות בתוך הגדרת כל אופרטור).

I הוא מצב אחד מקבוצת המצבים S כפי שהגדרנו אותה למעלה כך שהמיקומים מוגרלים בכל יצירת סביבה, שתי החבילות לא נאספו, לכל סוכן 20 יחידות הטענה ו0 קרדיטים.

G קבוצת כל המצבים בהם *T* =0 (T מסמל את הזמן שנותר לשחק).

2. **(יבש: 4 נק') הגדירו היוריסטיקה משלכם להערכת מצבי המשחק. עליכם לתעד אותה בנוסחה מפורשת ועלייה לכלול לפחות שלושה מאפיינים של הסביבה. בחרו שמות ברורים בנוסחה שלכם.**

pickUpIncentive = (robot.package is None) *
min(dist(robot.position,package_0.position,(dist(robot.position,package_1.position))

תמריץ לאיסוף חבילה. במצב בו אין חבילה רכיב זה שווה למרחק מנהטן מהחבילה הקרובה ביותר.

closestCharge = argmin{dist(robot.position, charge_0.position) , dist(robot.position, charge_1.position)

קביעת תחנת הטעינה הקרובה ביותר לסוכן.

closestPackageToCharge = argmin{dist(closestCharge.position, package_0.position), dist(closestCharge.position, package_1.position)}

קביעת החבילה הקרובה ביותר לתחנת הטעינה הקרובה ביותר.

after_charge = dist (closestCharge.position,closestPackageToCharge.position) +

dist(closestPackageToCharge.position, closestPackageToCharge.destination)

המרחק שנצטרך לעבור מתחנת הטעינה ועד מסירת חבילה בהנחה שאין לסוכן כרגע חבילה.

after_charge = dist (closestCharge.position, robot.package.destination)

אם הסוכן כן מחזיק בחבילה, הפונקציה מחזירה את המרחק שנצטרך לעבור מתחנת הטעינה ועד ליעד החבילה.

dropOffIncentive = (robot.package is not None) * dist(robot.position, robot.package.destination)

תמריץ למסירת החבילה, במצב בו יש לסוכן חבילה זהו מרחק מנהטן מיעד המסירה שלה, אחרת אפס.

אם מתקיים התנאי הבא לפיו הסוללה חלשה והטענה תסייע לביצוע מסירה נוספת, כלומר ברצוננו שהסוכן יבצע טעינה:

robot.battery <= 1.5 * dist(robot.position, closestCharge.position) and

after_charge >= robot.battery + robot.credit

אזי ערך היוריסטיקה:

H(s,agent) = -dist(robot.position, closestCharge.position) + 50 * (robot.package is not None)

אחרת (כלומר במצב שהסוכן לא זקוק להטענה):

H(s,agent) = -(15 * pickUpIncentive + dropOffIncentive - (100 * robot.battery)* (after_charge >= robot.battery + robot.credit) – (50 * robot.credit) + 50 * (robot.package is not None) + robot.position.Yaxis

יורסיטקה שכוללת את התמריצים שחישבנו ועוד. מיקום ה y של הרובוט משמש כשובר שיוויון במצבים בהם שני תאים באותה עמודה בעלי ערך יוריסטי זהה.

3. **(רטוב: 10 נק') ממשו בקובץ submission.py את הפונקציה `smart_heuristic` שבה משתמש הסוכן AgentGreedyImproved**

4. **(יבש: 2 נק') מהו החיסרון העיקרי של האלגוריתם? (לעומת minimax)**

האלגוריתם החמדני שכתבנו פועל בכל פעם רק לפי המצב העוקב לו, ולא לוקח בחשבון פעולות עתידיות של השחקן היריב. ייתכן שפעולה תהיה לטובתנו או לרעתנו כתלות בהתנהגות השחקן היריב. אלגוריתם לדוגמת minimax ייקח זאת בחשבון בעוד שאלגוריתם חמדני יבחן את הפעולה רק בהינתן המיקום הנוכחי של היריב.

חלק ב - **RB-Minimax**

1. **(יבש 3 נק') מה היתרונות והחסרונות של שימוש בהיוריסטיקה קלה לחישוב לעומת היוריסטיקה קשה לחישוב בהינתן שהיוריסטיקה הקשה לחישוב יותר מיודעת מהקלה לחישוב ? בהינתן שאנו בmin-max מוגבל משאבים.**

<u>יתרונות</u>: היורסטיקה קלה לחישוב לוקחת פחות זמן, לכן יתאפשר לנו להגדיר את עומק העץ הנפרש המקסימלי לערך גדול יותר. דבר זה יאפשר לנו להסתכל על יותר צעדים קדימה. בכך אנחנו מעדיפים "כמות על איכות".

<u>חסרונות</u>: היוריסטיקה קלה לחישוב פחות מיודעת, לכן הערך היוריסטי לפיו נקבל החלטות עלול להיות פחות מהימן.

2. **(יבש: 4 נק') חברתכם לקורס דנה מימשה סוכן minimax, היא שמה לב כי לעיתים הסוכן יכול לנצח בצעד אחד אך הוא בוחר בצעד אחר. האם יש לה באג באלגוריתם? אם אין באג הסבירו מה באלגוריתם גורם להתנהגות שכזו. אם יש באג מה הוא יכול להיות?**

אין בהכרח באג באלגוריתם, מצב לדוגמה שגורם להתנהגות כזאת: במשחק עם ערכי ניצחון שונים (כל שחקן שואף לנצח עם הניקוד הגבוה ביותר), ייתכן שהסוכן נמצא במצב עם שני בנים. האחד מצב סופי של ניצחון עם ניקוד a כלשהו, והשני אינו סופי אך מבטיח שבתת העץ שלו מגיעים לניצחון עם ניקוד של לפחות b>a.

3. **(רטוב: 10 נק') עליכם לממש את המחלקה AgentMinimax בקובץ submission.py. שימו לב! הסוכן מוגבל משאבים, כאשר המשתנה time_limit מגביל את מספר השניות שהסוכן יכול לרוץ לפני שיחזיר תשובה. (הגבלת הזמן עלייה אתם נבדקים הינה שנייה כלומר 1 t-).**

4. (יבש: 3 נק') נניח שבסביבה היו K שחקנים במקום 2. אילו שינויים יהיה צריך לעשות במימוש סוכן Minimax? כתבו פסאודו קוד בדומה לזה שראינו בתרגול.

a. בהינתן שכל סוכן רוצה לנצח ולא אכפת לו רק ממכם.

```
Function Minimax(State, Agent):
        If G(State) the return U(State) OR D=0 then return h(State)
        Turn ← Turn(State)
        Children ← Succ(State)
        CurMax ← -∞
        Loop for c in children
                v←Minimax(c, Agent.next())
                if v[Agent.id] > CurMax[Agent.id]
                        CurMax ← v
        Return(CurMax)
```

השינויים שביצענו הם כדלהלן:

- כעת, הפונקציה minimax, היוריסטיקה, וה-utility כולן מחזירות tuple שמכיל את הערכים המתאימים לכל אחד מהסוכנים. זאת, משום שכעת לא ניתן להסיק מה הערך המתאים לכל סוכן מתוך ערך יחיד (במקור אפשר היה להכפיל במינוס 1).
- בהתאם לשינוי זה, הפונקציות של היוריסטיקה וה utility מקבלות כארגומנט רק את המצב הנוכחי ולא צריכות לקבל את הסוכן, שכן הן מחזירות בכל פעם את הערך היוריסטי/utility עבור K הסוכנים.
- ההנחה היא שכל שחקן בתורו מנסה למקסם את הערך הספציפי שמובטח לו מתוך ה-tuple, וכך למעשה השחקנים פועלים בצורה רציונלית ומשפרים את המצב של עצמם.

b. בהנחה והדבר היחיד שכל סוכן רוצה הוא שלא תנצחו.

```
Function Minimax(State, Agent):
    If G(State) the return U(State, Agent) OR D=0 then return h(State, Agent)
    Turn ← Turn(State)
    Children ← Succ(State)
    If Turn = Agent then:
        CurMax ← −∞
        Loop for c in Children
            v ←Minimax(c, Agent ,D-1)
            CurMax←Max(v, CurMax)
        Return(CurMax)
    else: (Turm ≠ Agent)
        CurMin ← ∞
        Loop for c in Children:
            v ←Minimax(c, Agent ,D-1)
            CurMin←Min(v, CurMin)
        Return(CurMin)
```

הפסאודו-קוד נשאר זהה לזה שראינו בתרגול, שכן במצב זה הסוכן שלנו מנסה למקסם את עצמו, וכל שאר הסוכנים מנסים למנמל את הסוכן שלנו בלבד (הדבר מתבטא בכך שפונקצית ה- utility והיוריסטיקה נקראת עבור הסוכן שלנו בלבד).

c. בהנחה שכל סוכן רוצה שהסוכן שאחריו בתור ינצח.

```
Function Minimax(State, Agent):
    If G(State) the return U(State) OR D=0 then return h(State)
    Turn ← Turn(State)
    Children ← Succ(State)
    CurMax ← -∞
    Loop for c in children
        v←Minimax(c, Agent.next())
        if v[Agent.next().id] > CurMax[Agent.next().id]
            CurMax ← v
    Return(CurMax)
```

הפתרון עבור מצב זה דומה מאוד לפתרון עבור סעיף א, ההבדל הוא שכל סוכן ממקסם את הערך בטאפל שמתאים לסוכן שאחריו ולא לעצמו.

חלק ג - **Alpha-Beta**

1. **(רטוב: 10 נק') ממשו שחקן אלפא - בטא מוגבל משאבים במחלקה AgentAlphaBeta בקובץ submission.py, כך שיתבצע גיזום כפי שנלמד בהרצאות ובתרגולים.**

2. **(יבש: 3 נק') האם הסוכן שמימשתם בחלק זה יתנהג שונה מהסוכן שמימשתם בחלק ב מבחינת זמן ריצה ובחירת מהלכים ? הסבירו.**

הסוכן יתנהג שונה, שכן בזכות הגיזום הוא מסוגל במגבלת הזמן להגיע עמוק יותר בעץ (בערך פי 2 יותר עמוק). בעקבות כך, הוא יבצע החלטות יותר מיודעות ואמור לספק ביצועים טובים יותר.

חלק ד - **Expectimax**

1. **(יבש: 3 נק') בהנחה ואתם משתמשים באלגרותים Expectimax נגד סוכן שמשחק באופן רנדומלי לחלוטין באיזה הסתברות תשתמשו? ומדעו?**

עבור סוכן שמשחק באופן רנדומלי לחלוטין נשתמש בהסתברות אחידה (במשחק המחסנים, p יהיה כמות 1 חלקי ה legal successors.

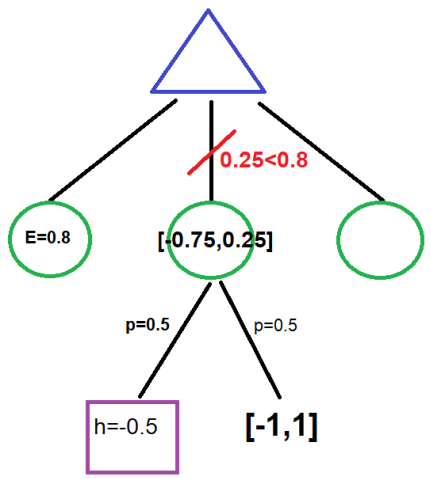
2. **(יבש: 4 נק') עבור משחקים הסתברותיים כמו שש בש, בהם יש מגבלת משאבים, משתמשים באלגוריתם RB-Expectimax. הניחו כי ידוע שהפונקציה היוריסטית h באלגוריתם Expectimax-RB מקיימת $\forall s: -1 \leq h(s) \leq 1$**
איך ניתן לבצע גיזום לאלגוריתם זה? תארו בצורה מפורטת את תנאי הגזימה, והסבירו את הרעיון מאחוריו.

נשים לב כי בזכות העובדה שהפונקציה היוריסטית חסומה בטווח [-1,1], ניתן להציב חסמים לערך התוחלת של צומת ה chance גם מבלי לחשב את כל תתי העצים שלו.

עבור שכבת chance שמתחת לשכבת max:

גיזום של צומת chance יתבצע כאשר החסם העליון על התוחלת שלו קטן/שווה לערך או לחסם העליון של צומת chance אחר מבין "האחים" שלו שכבר חושבו.

עבור שכבת chance שאחרי שכבת min, האלגוריתם יפעל בצורה זהה עבור חסם תחתון.



3. (רטוב: 10 נק') הסוכן של minimax ו-alpha-beta מניח שהסוכן היריב יבחר באופרטור שיוביל לתוצאה האופטימלית בתורו, אולם זה לא תמיד מתרחש.

לדוגמה, כאשר אנו מתחרים עם סוכן חמדן, סביר להניח שהוא לא יבחר בפעולה האופטימלית בכל צעד. אפשר להתחשב באפשרות שהיריב יבחר בפעולה שאינה אופטימלית בתורו באמצעות סוכן Expectimax.

גילינו מידע סודי על הרובוט המתחרה, הוא בוחר בין כל הפעולות בצורה יוניפורמית (בצורה אחידה) אבל לתזוזה ימינה ולאסיפת חבילה (כאשר פעולות אלו אפשריות) יש הסתברות גדולה פי 2 מלשאר הפעולות.

ממשו אלגוריתם Expectimax המשתמש במידע הסודי שקיבלתם.

חלק ה - משחק עם פקטור סיעוף גדול

1. **(יבש: 6 נק') להלן שינויים אפשריים ששוקלים בינה מחסנים לעשות במשחק בכדי לבחון יכולות נוספות של הרובוטים. עבור כל שינוי ציינו מה ההשפעה שלו על מקדם הסיעוף וחשבו את מקדם הסיעוף החדש המתקבל.**
   a. **הגדלת לוח המשחק להיות 8X8 והוספת מחסומים בסביבה. (מחסומים משמע משבצות שהסוכן לא יכול לעבור בהן)**

   בדיוק כמו במצב הרגיל, יש עדיין 7 פעולות אפשריות ולכן מקדם הסיעוף נשאר 7.
   b. **הוספת היכולת של רובוט בכל תור לבחור משבצת על הלוח ולהניח עלייה בלוק, משמע בכל תור יכול הרובוט לנוע למעלה, למטה, ימינה, שמאלה, לאסוף חבילה, להוריד חבילה, להטעין, ולהניח בלוק על הלוח, בלוק יכול להיות מונח על כל משבצת ריקה.**

   כעת נוספו פעולות חדשות. מדובר ב 25=5x5 פעולות חדשות, אך נשים לב כי בכל מצב של המשחק יש לכל היותר 21 אופציות לתאים ריקים (שכן בכל סיטואציה 4 תאים תפוסים ע"י תחנות טעינה ויעדי מסירת חבילה). לכן מקדם הסיעוף החדש הוא 21+7=28.

   (ניתן להתייחס לניסיון הנחת בלוק על משבצת שאינה ריקה כפעולה שמחזירה שגיאה, ואז נתייחס למס' פעולות חדשות של 25, כלומר סה"כ 25+7=32)
2. **(יבש: 6 נק') בהנחה ומימשו את השינוי השני עבור הסביבה (סעיף b1)**
   a. **האם יש אלגוריתם מהסעיפים הקודמים שנוכל להשתמש בו שזמן הריצה שלו סביר? (סביר משמע לא גדול מהותית מהזמן שלוקח לו להחזיר צעד עבור המשחק בלי השינוי).**

   נוכל להשתמש באלגוריתם greedy עם היוריסטיקה החכמה שכתבנו, שכן הוספת הפעולות תוסיף כמות סופית וקבועה של חישובי היוריסטיקה.

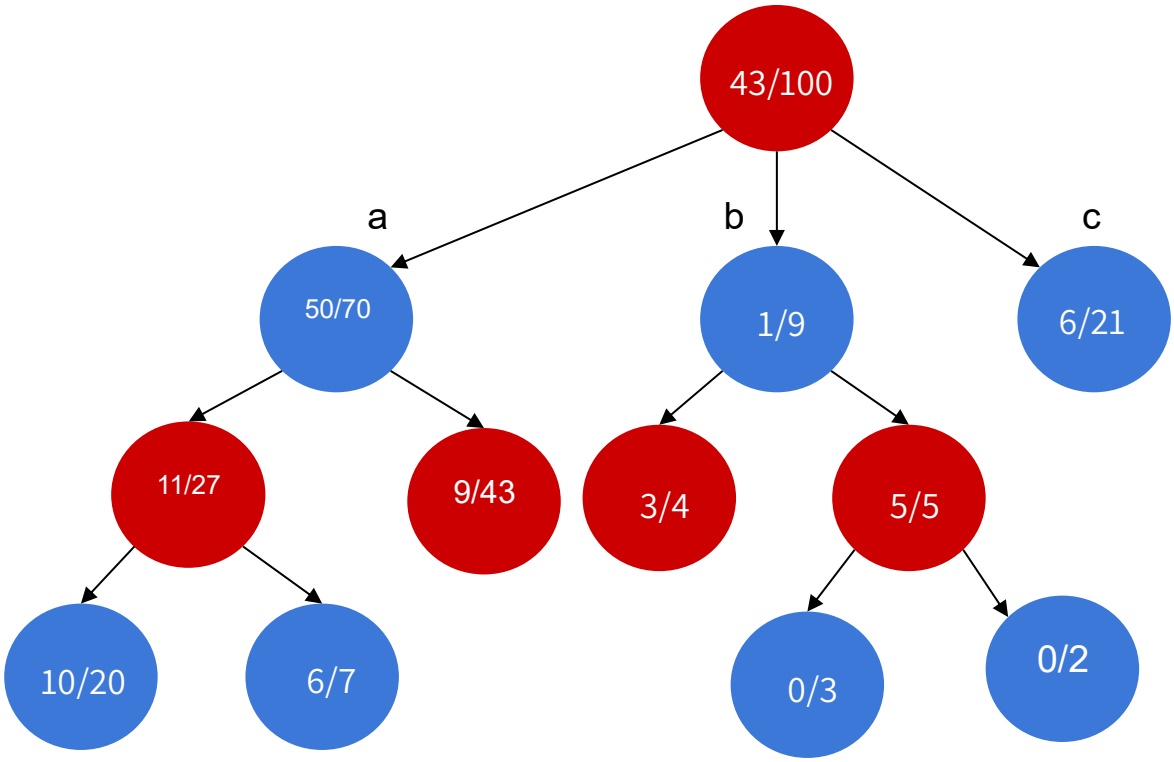
   לעומת זאת, בשאר האלגוריתמים ההוספה תגדיל את מקדם הסיעוף מה שישפיע בצורה רבה יותר על זמן החישוב, שכן אלגוריתמים אלה רצים בזמן אקספוננציאלי במקדם הסיעוף.
   b. **הציעו אלגוריתם שונה מאלו שממשתם בסעיפים הקודמים שנלמד בקורס שירוץ בזמן סביר. הסבירו מדוע בחרתם בו ולמה הוא טוב להתמודדות עם האתגר שנוצר משינוי הסביבה.**

   ניתן להשתמש באלגוריתם Monte-Carlo משום שכאשר מקדם הסיעוף עולה, אלגוריתמי minimax/expectimax רצים זמן רב יותר משום שכל עומק גדל בצורה מעריכית. לחילופין, תחת מגבלת זמן הם מגיעים לעומק קטן יותר משמעותית.

לעומת זאת, אלגוריתם Monte-Carlo מבצע rollouts שזמן ריצתם לא תלוי במקדם הסיעוף. כפי שלמדנו בהרצאה, אלגוריתם Monte-Carlo עובד בצורה טובה עבור משחקים עם מקדם סיעוף גדול, בהם minimax (ודומיו) לא יכול להגיע לעומק מספק על מנת לקבל החלטה אופטימלית.

חלק ו - יבש - שאלה פתוחה - **MCTS**

שחקן אדום ושחקן כחול שיחקו משחק. להלן עץ המשחק שמתאר את העץ שנוצר בשלב ביניים בהרצת MCTS עם פיתוח צמתים לפי 1UCB על משחק סכום אפס בין שניהם, נתון $C = \sqrt{2}$.



דגש על האלגוריתם: הערך בצומת הכחול מייצג את כמות הניצחונות של השחקן האדום מתוך כמות המשחקים שבוצעו עם הצעד הזה (ולהיפך). למשל צומת b מייצג את כמות הניצחונות של השחקן האדום אם בחר בפעולה שגרמה לו להגיע למצב b מתוך סך כל המשחקים ששוחקו עם הצעד.

1. **(5 נק') חלק מהערכים בצמתים נמחקו, השלימו את החלקים החסרים, אין צורך לנמק.**

2. **(5 נק') הצומת הבא שייבחר בשלב ה - selection יהיה (הוסיפו חישובים לנמק את בחירתכם):**

a. צאצא של a

$$\frac{50}{70} + \sqrt{2} \times \frac{\sqrt{\ln(100)}}{\sqrt{70}} = 1.08$$

b. צאצא של b

$$\frac{1}{9} + \sqrt{2} \times \frac{\sqrt{\ln(100)}}{\sqrt{9}} = 1.12$$

c. צאצא של c

$$\frac{6}{21} + \sqrt{2} \times \frac{\sqrt{\ln(100)}}{\sqrt{21}} = 0.95$$

3. **(5 נק')בהנחה שכל סימולציה מכאן והלאה מסתיימת בניצחון של השחקן הכחול מה מספר הניצחונות המינימלי שנדרש כדי שצאצא אחר של השורש ייבחר בשלב ה-selection (הוסיפו חישובים לנמק את תשובתכם)?**

לאחר rollout ראשון ב b:

$$UCB1(a) = \frac{50}{70} + \sqrt{2} \times \frac{\sqrt{\ln(100+1)}}{\sqrt{70}} = 1.077$$

$$UCB1(b) = \frac{1}{9+1} + \sqrt{2} \times \frac{\sqrt{\ln(100+1)}}{\sqrt{9+1}} = 1.061$$

$$UCB1(c) = \frac{6}{21} + \sqrt{2} \times \frac{\sqrt{\ln(100+1)}}{\sqrt{21}} = 0.95$$

1. **(3 נק') כעת רוצים לבצע שינוי כך שנעדיף exploration יותר מ - exploitation. הגישה לנוסחה שמחשבת את ה - 1UCB חסומה לכם, אך הנוסחה משתמשת ב - $N(s)$ אשר אליו יש לכם גישה ואתם יכולים לשנותו. כיצד תשנו אותו בכדי שהנוסחה החדשה שתיווצר תעדיף יותר exploration מ - exploitation לעומת הנוסחה הקודמת.**

נבחר קבוע $\alpha > 1$, לדוגמה 2, ונגדיר כעת את N(s) להיות מס' ה-rollouts שעברו דרך s כפול α. את החלק של ה-explotation פעולה זו תקטין ב-α. בניגוד לכך, את החלק של ה-exploration היא תקטין רק ב $\sqrt{\alpha}$ (בחירת ה $\alpha > 1$ מבטיחה ש $\sqrt{\alpha} < \alpha$) ותגדיל את ביטוי ה- ln בשיעור מסוים. סה"כ קיבלנו שכעת ה-exploration בעל יותר משקל.