אלגוריתמים כלכליים מטלה 9 – שי גאלי

קישור לקוד:

https://gist.github.com/ShayGali/37e5a1a06d8e6552384b717814c1fa7e

* שאלה 1: דיקטטורה סדרתית אקראית (random serial dictatorship)

רוצים לחלק n מטלות בין n שחקנים, כך שכל שחקן יקבל מטלה אחת בדיוק. אלגוריתם **דיקטטורה סדרתית אקראית** פועל באופן הבא:

- מסדרים את השחקנים בסדר אקראי כלשהו;
- כל שחקן בתורו בוחר את המטלה שהוא הכי רוצה, מבין המטלות שנשארו.

הוכיחו שהאלגוריתם מקיים את התכונות הבאות:

א. מגלה-אמת;

ב. יעיל בדיעבד (בהנחה שעבור כל שחקן, כל המטלות הן שונות – אף שחקן אינו אדיש בין שתי מטלות).

ג. **לא** יעיל לכתחילה;

ד. פרופורציונלי לכתחילה;

ה. **לא** ללא-קנאה לכתחילה;

ו. **לא** פרופורציונלי בדיעבד.

[א, ב, ד, ו קלים יחסית; ג, ה יותר קשים]

א. ברור שהאלגוריתם מגלה אמת – זה ממש כמו האלגוריתם שלך המכרז הראשון כאשר יש לנו n אנשים ו n חפצים, ראינו במטלה 5 שלא קיימת במקרה כזה התחכמות בטוחה, וגם כאן, לשחקן לא משתלם להגיד ערך אחר ממה שהוא מעריך – אם הוא יוריד מתחת למטלה אחרת – הוא יפסיד את המטלה שלו בתורו, אם הוא יגיד יותר, לא ייצא זה כלום, לכן לא משתלם לו להגיד שקר.

ב. ניזכר בהגדרה:

הגדרה. חלוקה אקראית נקראת **יעילה בדיעבד (ex-post efficient)** אם כל תוצאה של ההגרלה היא חלוקה יעילה־פארטו: בכל תוצאה של ההגרלה, לא קיימת חלוקה אחרת, הנותנת לשחקן אחד ערך גבוה יותר ("זה נהנה"), ולכל שאר השחקנים לפחות אותו ערך ("זה לא חסר").

הוכחה:

אחרי שחילקנו כבר את המטלות נשים לב שאין שיפור פארטו של החלוקה – שיפור פארטו במקרה הזה יהיה להחליף את המטלה של הבן האדם ה i בסדר לבן אדם ה j.
נניח בה"כ ש i היה לפני j, אז i רצה את המטלה שלו יותר מהמטלה של j, לכן אם נחליף, בטוח i ייפגע (כי הוא אינו אדיש בין המטלות), ולכן זה לא קיים שיפור.

ג. ניזכר בהגדרה:

חלוקה אקראית נקראת **יעילה לכתחילה (ex-ante efficient)** אם לא קיימת חלוקה אקראית שהיא שיפור־פארטו־לכתחילה שלה.

נראה דוגמא שהיא לא יעילה לכתחילה בה:

כדי לחשב זאת ניעזר בקוד הבא:

פונקציה שמקבלת את הערכים, משימות וסידור, ומחשב תוחלת

```python
import itertools
from typing import List,Dict

def clac_expected_value(players_to_values:Dict[str, Dict[str,int]], tasks:List[str],players_permutations:List[str]) -> Dict[str, float]:
    expected_values = {player: 0 for player in players_to_values.keys()}
    for perm in players_permutations:
        remaining_tasks = tasks.copy()
        for player in perm:
            # get the best task for the player
            best_task = max(remaining_tasks, key=lambda task: players_to_values[player][task])
            expected_values[player] += players_to_values[player][best_task]
            remaining_tasks.remove(best_task)

    for player in expected_values.keys():
        expected_values[player] /= len(players_permutations)

    return expected_values
```

שימוש:

```python
def method2_advantage_example():
    players_to_values = {
        "1": {"a": 1, "b": 1, "c": 1},
        "2": {"a": 75, "b": 4, "c": 25},
        "3": {"a": 60, "b": 65, "c": 5},
    }

    tasks = ["a", "b", "c"]

    print("Method 1 (all permutations):")
    # All 6 possible permutations
    players_permutations = list(itertools.permutations(players_to_values.keys()))
    expected_values = calc_expected_value(
        players_to_values, tasks, players_permutations
    )
    print("\tExpected values:")
    for player, value in expected_values.items():
        print(f"\t\tPlayer {player}: {value}")

    print("\nMethod 2 (strategic permutations):")
    # Only include permutations that lead to optimal assignments
    players_permutations = [
        ["3", "2", "1"],
        ["3", "1", "2"],
    ]
    expected_values = calc_expected_value(
        players_to_values, tasks, players_permutations
    )
    print("\tExpected values:")
    for player, value in expected_values.items():
        print(f"\t\tPlayer {player}: {value}")
```

פעם אחת (method1) זה כל הסידורים (ככה נחשב את התוחלת)

פעם שניה לפי נגדיר שיש רק שני סידורים אחרים, כל אחת בהסתברות חצי.

הפלט שנקבל:

```
Method 1 (all permutations):
        Expected values:
                Player 1: 0.9999999999999999
                Player 2: 50.0
                Player 3: 55.00000000000001

Method 2 (strategic permutations):
        Expected values:
                Player 1: 1.0
                Player 2: 50.0
                Player 3: 65.0
```

כלומר קיבלנו שהתוחלת של 3 עלתה, ושל שחקנים 1,2 זה לא שינה – לכן החלוקה שלנו לא הייתה יעילה לכתחילה.

ד. ניזכר בהגדרה:

הגדרה. חלוקה אקראית נקראת:

- **פרופורציונלית לכתחילה (ex-ante proportional)** – אם תוחלת הערך של כל שחקן היא לפחות 1/n מהערך שהוא מייחס לכל החפצים:

$$E[v_i(x_i)] \geq v_i(All)$$

נשים לב שהמקרה כי גרוע עבור השחקן ה i שאם הוא בוחר בתור ה j הוא יקבל את המטלה שנמצאת במקום ה j אצלו – כלומר כל השחקנים לפניו "גנבו" לו את המטלות שהוא רצה, והוא נשאר רק עם ה j.
לכן התוחלת של המקרה הכללי (שיכול להיות שלא "יגנבו" לו) תהיה גבוה יותר מאשר המקרה הפרטי. נשים לב שהמיקום של השחקן ה i נקבע באופן אקראי ואחיד ולכן,:

נסדר את ההעדפות של השחקן ה i, ונסמן את הסדר המסודר $t_1, \dots, t_n$, ואז

$$E[V_i(X_i)] \geq \frac{1}{n}V(t_1) + \cdots + \frac{1}{n}V(t_n) = \frac{v_i(All)}{n}$$

כאשר האיבר הראשון זה ההסתברות שהוא ייבחר ראשון כפול מה שהוא יקבל, ..., האיבר ה n זה ההסתברות שהוא יבחר האחרון כפול מה שהוא יקבל.
בגלל שיהיו שם כל המטלות, נקבל שזה הממוצע של הערכים על הכל.

ה. ניזכר בהגדרה:

- **ללא־קנאה לכתחילה (ex-ante envy-free)** – אם לכל שני שחקנים i, j, תוחלת הערך של שחקן i בעיני עצמו גדולה לפחות כמו תוחלת הערך של שחקן j בעיני שחקן i:

$$E[v_i(x_i)] \geq E[v_i(x_j)]$$

נראה דוגמא שקיימת קנאה לכתחילה, גם כאן נעזרי בקוד:

```python
def calc_full_expected_value(players_to_values:Dict[str, Dict[str,int]], tasks:List[str]) -> Dict[str, Dict[str, float]]:

    expected_values = {
        player: {p: 0 for p in players_to_values.keys()}
        for player in players_to_values.keys()
    }
    players_permutations = list(itertools.permutations(players_to_values.keys()))
    print(f"players_permutations: {players_permutations}")
    for perm in players_permutations:
        remaining_tasks = tasks.copy()
        for player in perm:
            # get the best task for the player
            best_task = max(
                remaining_tasks, key=lambda task: players_to_values[player][task]
            )
            remaining_tasks.remove(best_task)
            for p in perm:
                # update the expected values of all of the players in the perspective of the player
                expected_values[p][player] += (players_to_values[p][best_task]) / len(players_permutations)


    return expected_values
```

הקריאה + פלט:

```python
def not_envy():
    players_to_values = {
        "p1": {"a": 0,  "b": 5, "c": 0},
        "p2": {"a":10,  "b": 0, "c": 4},
        "p3": {"a": 5,  "b": 4, "c": 0},
    }
    tasks = ["a", "b", "c"]
    expected = calc_full_expected_value(players_to_values, tasks)
    print("Expected values:")
    for player, value in expected.items():
        print(f"\tPlayer {player}:")
        for p, v in value.items():
            print(f"\t\tPlayer {p}: {v}")
```

```
Expected values:
        Player p1:
                Player p1: 4.166666666666667
                Player p2: 0.0
                Player p3: 0.8333333333333334
        Player p2:
                Player p1: 0.6666666666666666
                Player p2: 7.000000000000001
                Player p3: 6.333333333333334
        Player p3:
                Player p1: 3.333333333333333
                Player p2: 2.5
                Player p3: 3.166666666666667
```

כלומר שחקן 3 יקנא בשחקן 1 לכתחילה!

ו. ניזכר בהגדרה:

הגדרה. חלוקה אקראית נקראת:

* **פרופורציונלית בדיעבד (ex-post proportional)** – אם בכל תוצאה של ההגרלה, הערך של כל שחקן הוא לפחות $1/n$ מהערך שהוא מייחס לכל החפצים.

נראה דוגמא שלא מקיימת. ניקח את הדוגמא מההרצאה

אלי: משמרת יום: 10-, משמרת לילה: 20-

בני: משמרת יום: 20-, משמרת לילה: 10-

נניח בני קיבל את היום ואלי את הלילה, אז

$$V(X_1) = -20 \leq -\frac{30}{2}$$

ולכן היא לא פרופורציונלית בדיעבד.

גילוי נאות: את הקוד כתבתי בvscode כאשר copilot מותקן. אני חייב לציין שהוא כותב על הפנים כשמדובר על אלגוריתמים שצריך לחשוב טיפה, כמו אלו.