# חלוקה אגליטרית של חפצים בדידים

כשהחפצים העומדים לחלוקה הם בדידים, אנחנו לא יכולים להבטיח חלוקה פרופורציונלית, אבל אנחנו יכולים לנסות להתקרב לחלוקה כזאת כמיטב יכולתנו במסגרת נתוני הבעיה. באחד השיעורים הקודמים למדנו על עקרון המתאים לבעיה זו – העקרון האגליטרי. כזכור, חלוקה אגליטרית היא חלוקה שבה הערך המינימלי של שחקן גדול לפחות כמו הערך המינימלי של שחקן בכל חלוקה אחרת.

כשהחפצים היו רציפים, יכולנו לחשב חלוקה אגליטרית במהירות ע"י פתרון בעיית אופטימיזציה קמורה. אבל כשהחפצים בדידים, הבעיה הרבה יותר קשה.

**משפט**. מציאת חלוקה אגליטרית היא בעיה NP-קשה אפילו כשיש רק שני שחקנים עם פונקציות-ערך זהות.

**הוכחה**. ההוכחה היא על-ידי רדוקציה מהבעיה Partition – בעיה שידוע שהיא NP-קשה (ראו כאן: https://www.ics.uci.edu/~goodrich/teach/cs162/notes/pnp3.pdf ). בבעיה זו, נתונים לנו מספרים שלמים חיוביים $a_1,\dots,a_m$, והמטרה היא להחליט, האם קיימת חלוקה של המספרים לשתי קבוצות, כך שהסכום בשתי הקבוצות הוא זהה.

בהינתן בעיית Partition, נבנה בעיית חלוקה אגליטרית באופן הבא:

- יש שני שחקנים עם פונקציות-ערך זהות.
- יש m חפצים. כל שחקן מייחס לחפץ ה-j ערך $a_j$.

נמצא חלוקה אגליטרית ונבדוק את הערך המינימלי שמקבל שחקן בחלוקה זו. אם הערך המינימלי הוא בדיוק חצי מסכום כל הערכים ($0.5(a_1+\dots+a_m)$), אז יש לנו חלוקה שווה, והתשובה לבעיית ה-Partition היא "כן"; אחרת, הערך המינימלי קטן יותר מחצי סכום הערכים, ולא קיימת חלוקה שבה הערך המינימלי שווה חצי מסכום הערכים, ולכן והתשובה לבעיית ה-Partition היא "לא". ***

מה עושים כשנתקלים בבעיה NP-קשה? יש כמה אפשרויות.

## א. פתרון בעיית אופטימיזציה במספרים שלמים

אפשר לנסות לפתור את הבעיה הקשה על-ידי כתיבת **בעיית אופטימיזציה במספרים שלמים**. אנחנו יכולים לכתוב בעיה דומה לזו שכתבנו כדי למצוא חלוקה אגליטרית של משאבים רציפים, אבל להוסיף אילוץ הקובע שהמשתנים חייבים להיות שלמים. נדגים על אותה דוגמה שראינו בשיעור 2. יש שני שחקנים ושלושה חפצים עם הערכים הבאים:

| **חפץ:** | **שרשרת** | **טבעת** | **צמיד** |
|---|---|---|---|
| **עמי:** | 1 | 19 | 80 |
| **תמי:** | 29 | 1 | 70 |

פתרון אפשרי בפייתון הוא:

```
import cvxpy
x, y, z = cvxpy.Variable(3, integer=True)
```

```
min_utility = cvxpy.Variable()
utility_ami = x*80 + y*19 + z*1
utility_tami = (1-x)*70 + (1-y)*1 + (1-z)*29

prob = cvxpy.Problem(
    cvxpy.Maximize(min_utility),
    constraints = [0 <= x, x <= 1, 0 <= y, y <= 1, 0 <= z, z <=
1, min_utility <= utility_ami, min_utility <= utility_tami])
prob.solve()
print("status:", prob.status)
print("optimal value: ", prob.value)
print("Fractions given to Ami: ", x.value, y.value, z.value)
print("Utility of Ami", utility_ami.value)
print("Utility of Tami", utility_tami.value)
```

זה בדיוק אותו קוד שהשתמשנו בו עבור משאבים רציפים, פרט לעובדה שאנחנו מגדירים את המשתנים כשלמים (integer=True).

כאשר מספר השחקנים והחפצים מספיק קטן, יש סיכוי טוב שנקבל פתרון מהיר. אבל כיוון שהבעיה היא NP-קשה, במקרה הגרוע הפתרון ידרוש זמן מעריכי בגודל הבעיה. כאשר מספר השחקנים והחפצים הוא גדול, ייתכן שהפתרון לא יימצא בזמן סביר.

## ב. אלגוריתמי קירוב

דרך נוספת להתמודד עם בעיה NP-קשה היא לפתח **אלגוריתמי קירוב –** אלגוריתמים הנותנים פתרון קרוב לפתרון הטוב ביותר. במקרה שלנו, כיוון שהמטרה היא למקסם את הערך האגליטרי (הקטן ביותר), אנחנו מחפשים אלגוריתם המחזיר, עבור כל קלט, חלוקה שבה הערך האגליטרי הוא לפחות r כפול הערך האגליטרי האופטימלי, כאשר r הוא מספר ממשי כלשהו.

גם למצוא אלגוריתם-קירוב זה לא פשוט כלל. אנחנו נראה אלגוריתם קירוב עבור מקרה פרטי שבו לכל השחקנים יש אותה פונקציית-ערך – הם מסכימים על הערכים של כל החפצים. זה מצב נפוץ, למשל, כאשר הערכים הם מחירי-שוק, והמטרה היא למצוא חלוקה אגליטרית על-פי מחירי השוק.

אלגוריתמי-הקירוב הראשונים לבעיה זו פותחו עבור בעיה הנדסית, שבמבט ראשון אין כל קשר בינה לבין הגינות: בעיית **תיזמון תהליכים במחשב מרובה-מעבדים** (באנגלית: multiprocessor scheduling; נקראת גם machine scheduling או job scheduling). יש לנו מחשב עם כמה מעבדים זהים; אנחנו רוצים לבצע בעזרתו משימה חישובית מסויימת, המורכבת ממספר רב של תהליכים. אנחנו יודעים מראש כמה זמן דרוש למעבד לבצע כל תהליך. השאלה היא, איך לחלק את התהליכים בין המעבדים, כך שהמשימה תסתיים בזמן הקצר ביותר האפשרי?

**דוגמה א**. יש ארבעה מעבדים. המשימה מורכבת מתשעה תהליכים. אורכי התהליכים בשניות הם:

7, 7, 6, 6, 5, 5, 4, 4, 4

נבדוק שתי חלוקות אפשריות של תהליכים למעבדים:

- חלוקה א: 4+4+7, 4+7, 5+6, 5+6. שלושה מעבדים מסיימים תוך 11 שניות, אבל המעבד הרביעי מסיים תוך 15 שניות, ולכן זמן הסיום של המשימה כולה הוא 15.
- חלוקה ב: 4+4+4, 5+7, 5+7, 6+6. כל המעבדים מסיימים תוך 12 שניות, ולכן זמן הסיום של המשימה כולה הוא 12. זוהי החלוקה היעילה ביותר בדוגמה זו. ***

הסיבה הראשונה לעיסוק בבעיה זו היתה, כאמור, למצוא חלוקת-עבודה יעילה ומהירה. אך למעשה, בעיה זו שקולה לחלוטין לבעייה של חלוקה אגליטרית של חפצים עם ערך שלילי בין שחקנים עם הערכות זהות.

דוגמה לחפצים בעלי ערך שלילי היא: מטלות, כגון תורנויות בבית או בבסיס. רוב האנשים לא אוהבים לבצע מטלות, אבל הערך (השלילי) שהם מייחסים לכל מטלה עשוי להיות שונה.

**דוגמה ב**. יש ארבעה שחקנים הצריכים לבצע ביחד תשע מטלות. הם מייחסים למטלות ערכים שליליים:

-7, -7, -6, -6, -5, -5, -4, -4, -4

(כולם מסכימים על הערכים של כל מטלה). אנחנו רוצים למצוא חלוקה אגליטרית. נבדוק שתי חלוקות:

- חלוקה א: שחקן אחד מקבל שלוש מטלות עם ערכים -7, -4, -4; שחקן שני מקבל שתי מטלות עם ערכים -7, -4; ועוד שני שחקנים מקבלים כל אחד שתי מטלות עם ערכים -5, -6. ערכי השחקנים הם: -15, -11, -11, -11.
- חלוקה ב: שחקן אחד מקבל שלוש מטלות עם ערכים -4, -4, -4; שחקן שני מקבל שתי מטלות עם ערכים -6, -6; ועוד שני שחקנים מקבלים כל אחד שתי מטלות עם ערכים -5, -7. ערכי השחקנים הם: -12, -12, -12, -12.

שימו לב שבחלוקה א הערך האגליטרי (- הקטן ביותר) הוא מינוס 15, ובחלוקה ב הערך האגליטרי הוא מינוס 12, שהוא מספר גדול יותר. חלוקה ב היא החלוקה האגליטרית במקרה זה. ***

קל לראות שהדבר נכון באופן כללי: כל חלוקת תהליכים למעבדים, הממזערת את זמן-הריצה המקסימלי, שקולה לחלוקת מטלות לאנשים, הממקסמת את הערך (השלילי) המינימלי – חלוקה אגליטרית. הבעיה היא NP-קשה – ההוכחה זהה להוכחה עבור חפצים עם ערכים חיוביים (למעלה).

אנחנו נראה אלגוריתם חמדני פשוט למציאת חלוקה אגליטרית מקורבת. לצורך הפשטות, כדי שנוכל לעבוד עם מספרים חיוביים, נניח שהקלט לבעיה הוא רשימת ערכי המטלות בערך מוחלט, והמטרה היא למצוא חלוקה שבה הערך המקסימלי הוא קטן ביותר.

1. סדר את המטלות בסדר יורד של הערך שלהם – מהגדול לקטן.
2. תן את המטלה הבא לשחקן שהערך הנוכחי שלו הוא קטן ביותר.
3. אם יש עוד מטלות, חזור לשורה 2.

בעולם תיזמון התהליכים, האלגוריתם הזה נקרא Longest Processing Time first, או בקיצור LPT, כי הוא מתזמן קודם את התהליכים שזמן-העיבוד שלהם ארוך ביותר. בעולם החלוקה ההוגנת, נקרא לו פשוט "האלגוריתם החמדני".

בדוגמה א למעלה, המטלה הראשונה בסדרה היא ה-7 והאחרונה היא ה-4. האלגוריתם נותן את ארבעת המטלות הראשונות לארבעת השחקנים בסדר כלשהו, למשל, א-7, ב-7, ג-6, ד-6. שתי המטלות הבאות – 5,5 – ניתנות לשחקנים שהערך הנוכחי שלהם הוא קטן ביותר, שהם ג ו-ד. שתי המטלות הבאות ניתנות לשחקנים שהערך הנוכחי שלהם הוא קטן ביותר, שהם א ו-ב. עכשיו, סכום הערכים של כל ארבעת השחקנים הוא 11, ולכן המטלה האחרונה ניתנת לשחקן כלשהו באקראי, נניח לשחקן א. בכל מקרה, ערכי השחקנים (בערך מוחלט) הם 15, 11, 11, 11, בדיוק כמו בדוגמה. אנחנו רואים שהאלגוריתם אינו מוצא את החלוקה האגליטרית (שהיא 12, 12, 12, 12), אבל הוא גם לא כל כך רחוק – הערך האגליטרי שלו גדול רק פי 1.25 מהערך האגליטרי האופטימלי.

כשמנתחים אלגוריתמי קירוב, התכונה המעניינת היא יחס הקירוב שלהם (approximation ratio) – היחס בין איכות הפתרון שהאלגוריתם מוצא, לבין איכות הפתרון האופטימלי. בבעיות מינימיזציה, כמו הבעיה שלנו, יחס הקירוב גדול מ-1, כי הפתרון של האלגוריתם בדרך-כלל גדול יותר מהפתרון האופטימלי; בבעיות מקסימיזציה, יחס הקירוב קטן מ-1, כי הפתרון של האלגוריתם בדרך-כלל קטן יותר מהפתרון האופטימלי. בדוגמה למעלה, יחס הקירוב של האלגוריתם החמדני היה 1.25=15/12. המשפט הבא מנתח את יחס הקירוב של האלגוריתם החמדני באופן כללי.

**משפט**. יחס הקירוב של האלגוריתם החמדני לחלוקת מטלות בין n שחקנים הוא לכל היותר:

$$4/3 - 1/(3n)$$

למשל, כאשר n=4, יחס הקירוב הוא 4/3-1/12 = 15/12 = 1.25 – בדיוק כמו בדוגמה למעלה. ניתן להכליל את הדוגמה ולהראות, עבור כל n, דוגמה שבה יחס-הקירוב הוא בדיוק הביטוי שבמשפט, כך שהמשפט הוא הדוק.

המשפט הוכח ע"י רונאלד גראהאם כבר בשנת 1969, בקשר לבעיית חלוקת-התהליכים. ההוכחה שלו היתה על-דרך השלילה; אנחנו נראה הוכחה פשוטה יותר, על-דרך החיוב.

**הוכחה**. נניח שהערך המקסימלי בחלוקה האופטימלית הוא OPT. לצורך ההוכחה, ננרמל את ערכי כל המטלות ע"י חלוקה ב-OPT. לאחר הנירמול, ערכי כל הסלים בחלוקה האופטימלית קטנים או שווים 1, ומכאן שערכי כל המטלות קטנים או שווים 1, וסכום ערכי כל המטלות קטן או שווה n. נחלק את המטלות לשני סוגים:

- מטלות שערכן גדול ממש מ-1/3 ייקראו גדולות;
- מטלות שערכן קטן או שווה ל-1/3 ייקראו קטנות.

מהנירמול נובע, שכל סל בחלוקה האופטימלית כולל לכל היותר שתי מטלות גדולות, ובסך-הכל יש לכל היותר 2n מטלות גדולות.

- בדוגמה א (שבה n=4), הנירמול מתבצע ע"י חלוקה ב-12. יש שש מטלות גדולות: 7/12, 7/12, 6/12, 6/12, 5/12, 5/12, והשאר קטנות (4/12). בחלוקה האופטימלית יש שלושה סלים עם שתי מטלות גדולות, ועוד סל אחד עם מטלות קטנות בלבד.

האלגוריתם החמדני, על-פי הגדרתו, מחלק קודם את כל המטלות הגדולות, ואז את כל המטלות הקטנות. אנחנו נוכיח את המשפט בשני טענות-עזר: טענה 1 מתייחסת לשלב חלוקת המטלות הגדולות, וטענה 2 מתייחסת לשלב חלוקת המטלות הקטנות.

טענה 1: לאחר שהאלגוריתם סיים לחלק מטלות גדולות, ערכי כל השחקנים קטנים או שווים 1.

הוכחה. אם מספר המטלות הגדולות הוא לכל היותר n, אז האלגוריתם החמדני נותן מטלה גדולה אחת בלבד לכל שחקן, וברור ושערכי כל השחקנים קטנים או שווים 1. לכן נניח שמספר המטלות הגדולות הוא n+t, עבור מספר שלם כלשהו t בין 1 ל-n.

נתאר לעצמנו גרף G שבו הקודקודים הם המטלות הגדולות, ושתי מטלות מחוברות בקשת אם-ורק-אם סכום ערכיהן קטן או שווה 1. כל סל בחלוקה האופטימלית מכיל לכל היותר שתי מטלות גדולות, שסכום ערכיהן קטן או שווה 1. כיוון שיש n+t מטלות גדולות, יש לכל היותר n-t סלים המכילים מטלה גדולה אחת או אפס. לכן בגרף G ישנו שידוך, שבו מספר המטלות הלא-משודכות הוא לכל היותר n-t. עכשיו נבדוק מה נובע מזה לגבי חיבורים בין מטלות בגרף (המטלות ממוספרות בסדר שבו האלגוריתם עובר עליהן, מהגדולה לקטנה):

- נתבונן במטלות 1 עד n-t+1. לפחות אחת מהמטלות האלו משודכת בחלוקה האופטימלית, ולכן לפחות מטלה אחת מחוברת למטלה אחרת כלשהי. לכן, המטלה הקטנה ביותר בקבוצה זו, שהיא מטלה n-t+1, בהכרח מחוברת למטלה הקטנה ביותר מבין הגדולות, שהיא מטלה n+t.

- נתבונן במטלות 1 עד n-t+2. לפחות שתיים מהמטלות האלו משודכות בחלוקה האופטימלית, ולכן לפחות שתי מטלות מקבוצה זו מחוברת לשתי מטלות אחרות כלשהן. לכן, המטלה הקטנה ביותר בקבוצה זו, שהיא מטלה n-t+2, בהכרח מחוברת למטלה השניה בקוטנה מבין הגדולות, שהיא מטלה n+t-1.
- באותו אופן, מטלה n-t+3 מחוברת למטלה n+t-2, ובאופן כללי, מטלה n-t+k מחוברת למטלה n+t-k+1 לכל k בין 1 ל-t. בפרט, מטלה n מחוברת למטלה n+1. המשמעות של "חיבור" זה היא, שסכום ערכי המטלות קטן או שווה 1.

כשהאלגוריתם החמדני מגיע למטלה n+1, הוא נותן אותה לשחקן שקיבל את המטלה הקטנה ביותר שחולקה עד כה, שהיא מטלה n; מההסבר למעלה נובע, שהערך הכולל של שחקן זה קטן או שווה 1. את מטלה n+2 הוא נותן לשחקן שקיבל את מטלה n-1, וכן הלאה; באופן כללי, את מטלה n+t-k+1 הוא נותן לשחקן שקיבל את מטלה n-t+k, וכפי שהסברנו למעלה, הערך הכולל של כל השחקנים האלה קטן או שווה 1.[1] ***

סיימנו את חלוקת המטלות הגדולות – עכשיו נראה מה קורה בשלב חלוקת המטלות הקטנות.

טענה 2: כאשר האלגוריתם נותן מטלה קטנה לשחקן כלשהו, ערכו החדש של השחקן (כולל המטלה שקיבל) קטן או שווה *4/3-1/3n*.

הוכחה. נבחין בשני מקרים:

- מקרה א: ערכו הישן של השחקן (לפני שקיבל את המטלה) קטן או שווה 1-1/3n. ערכה של המטלה החדשה הוא לכל היותר 1/3, ולכן ערכו החדש של השחקן לכל היותר 4/3-1/3n.
- מקרה ב: ערכו הישן של השחקן גדול מ- 1-1/3n. כיוון שהאלגוריתם בחר לתת את המטלה הבאה דווקא לשחקן זה, ערכם של השחקנים האחרים גדול לפחות באותה מידה. לכן, סכום ערכי n-1 השחקנים האחרים גדול מ:
- (n-1)-(n-1)/3n = n-4/3+1/3n
- אבל סכום ערכי המטלות כולן, כולל המטלה החדשה, הוא לכל היותר n. לכן, ערכו של השחקן ה-n לאחר שקיבל את המטלה החדשה הוא לכל היותר:
- n – (n-4/3+1/3n) = 4/3-1/3n.

משתי הטענות ביחד נובע המשפט. ***

## מקורות

- ויקיפדיה, Partition problem.
- ויקיפדיה, Longest-processing-time algorithm.
- טל גרינשפון, קורס "אלגוריתמי זימון ושיבוץ", אוניברסיטת אריאל.

סיכם: אראל סגל-הלוי.

---

1 מכאן גם נובע, שהאלגוריתם אינו נותן שלוש מטלות גדולות לאף שחקן, שכן הסכום של כל שלוש מטלות גדולות גדול ממש מ-1, והאלגוריתם החמדני תמיד נותן את המטלה הבאה למי שסכום ערכיו כולל מטלה זו יהיה הקטן ביותר.