בעיית תרמיל הגב
Knapsack problem

03.2022

נתן אפרתי

יצחק כהנא

מבוא

בעיית תרמיל הגב (Knapsack problem) היא בעיית אופטימיזציה ידועה הנחקרת בתחום מדעי המחשב.

בבעיה זו נתונה קבוצת עצמים שלכל אחד מהם משקל וערך, בנוסף נתון חסם על המשקל. המטרה היא למצוא את אוסף העצמים שערכם הכולל מרבי וסכום משקליהם אינו עולה על החסם הנתון.

בעיה זו מוכרת במדעי המחשב כבעיית NP שלמה, כלומר בעיה שלא ניתן לפתור אותה בזמן פולינומיאלי.

לבעית תרמיל הגב נמצאו שימושים רבים.
בין היתר שימשה הבעיה למערכת הצפנה בפרוטוקול שפותח על-ידי מרקל והלמן (ב-1978).

לבעיה זו 2 גרסאות:

1. גרסת 0-1, שבה צריך להחליט עבור כל פריט האם לקחתו או לא בשלמותו בלבד.
2. הגרסה השברית, בה עבור כל פריט ניתן לקחת גם חלק יחסי ממנו.

ישנו אלגוריתם חמדני פשוט העונה על בעיית הגרסה השברית בסיבוכיות $O(NLogN)$ ולכן לא נעסוק בו.

במסגרת פרויקט זה נדון בבעיה בגרסת 0-1.

הגדרה פורמלית

נתון תרמיל גב בעל קיבולת ידועה W.

ונתונים קבוצת עצמים I בגודל N כך שלכל עצם i נתונים:
משקל wi, וערך vi.
עבור כל עצם i נסמן ב xi את החלק מהשלם (או השלם כולו) מאותו עצם שנכניס לתרמיל.

המטרה היא למצוא אוסף של עצמים שערכם מרבי אך סכום משקליהם אינו עולה על החסם הנתון.

כלומר נרצה לחשב:

$$maximize \sum_{i-1}^{n} v_i x_i$$

$$subject\ to \sum_{i-1}^{n} w_i x_i \leq W\ and\ x_i \in \{0, 1\}$$

מטרות

לבחון מספר אלגוריתמים לפתרון הבעיה.

לבחון את ההבדלים בין אלגוריתמים שונים, מהו הדיוק שנקבל בזמן ריצה סביר.

הגישות לפתרון הבעיה

כאמור, הבעיה (בשלמותה) מוכרת כבעיית NP שלמה, ולא ניתן להגיע לפתרון מדויק בזמן פולינומיאלי.

ולכן יש מספר גישות לפתרון הבעיה.

1. חיפוש אחר פתרון מדויק - אפשרי בזמן אקספוננציאלי.
   או בזמן פולינומיאלי, אך ורק כאשר משקלי הפריטים והתיק הם שלמים.
2. חיפוש אחר קירוב לפתרון הבעיה.

נרחיב אודות האלגוריתמים האפשריים עבור הגישות הללו.

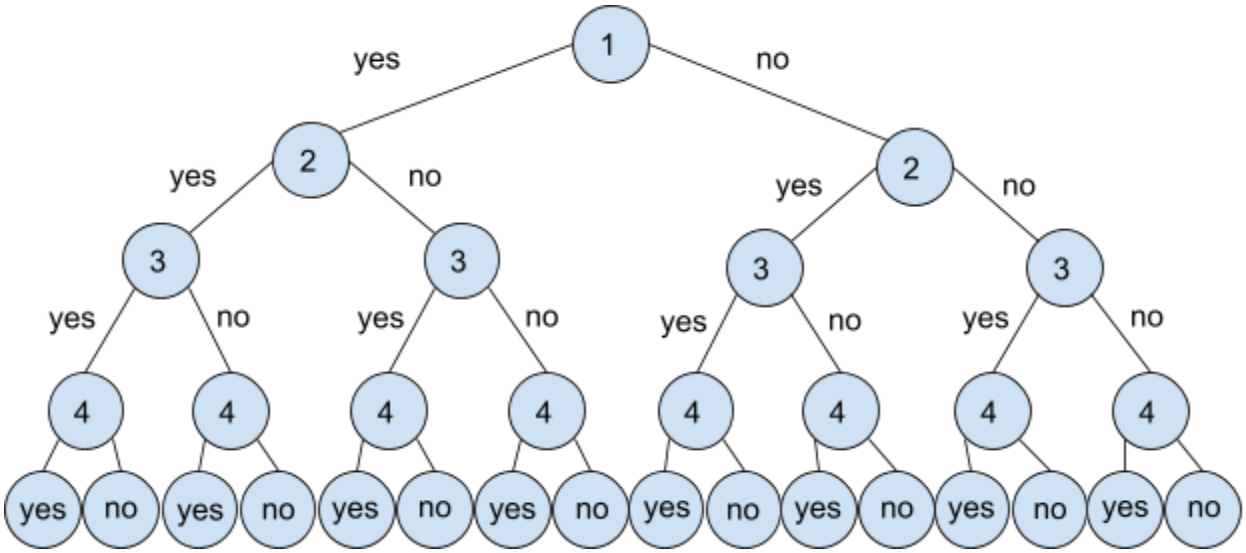
האלגוריתמים שנחקרו

1. האלגוריתם הנאיבי.
2. פתרון ע"י תכנון דינאמי.
3. האלגוריתם הגרידי (החמדני).
4. האלגוריתם הגנטי.

האלגוריתם הנאיבי:

אלגוריתם זה עובר על כל תתי הקבוצות מתוך קבוצת הפריטים ובוחר את תת הקבוצה בעלת הערך המקסימלי (כל עוד נכנסת בתיק).

דוגמא לגרף המתאר את המסלולים עבור ארבעה פריטים:



פתרון נאיבי רקורסיבי אפשרי:

בפתרון זה, W מייצג את גודל התיק, n הוא מספר הפריטים.
w הוא מערך המשקלים של הפריטים.
ו- v הוא מערך הערכים של הפריטים בהתאמה לסדר הפריטים במערך w.

```
def KS(n, W):
    if n == 0 or W == 0:                    -   אם עברנו על כל הפריטים אם שאין יותר מקום בתיק
        result = 0                          -   מסתיימת הרקורסיה
    else if w[n] > W:                       -   אם משקל הפריט גדול מהמשקל שנותר בתיק
        result = KS(n-1, W)                 -   נדלג על הפריט הנוכחי
    else:
        tmp1 = KS(n-1, W)                   -   חישוב האפשרות בלי לקיחת את הפריט הנוכחי
        tmp2 = v[n] + KS(n-1, W - w[n])     -   חישוב האפשרות עם לקיחת את הפריט הנוכחי
        Result = max { tmp1, tmp2 }         -   נבחר את המקסימום בין המקרים
    return result                           -   התוצאה הסופית (הערך המקסימלי)
```

כפי שניתן לראות, לאלגוריתם זה ישנה סיבוכיות זמן ריצה של $O(2^N)$ (לפחות כמספר תתי הקבוצות).
כלומר ניתן להגיע לפתרון מדויק אך בזמן ריצה אקספוננציאלי.

פתרון ע"י תכנון דינאמי:

פתרון זה נמנע מחישובים כפולים ע"י שמירת התוצאות (תת בעיות משותפות) בזיכרון.

פתרון זה מחייב שמשקל התיק ומשקלי הפריטים הם מספרים שלמים.

הסיבה לכך שהמשקלים חייבים להיות שלמים היא מכיון שאלגוריתם זה משתמש במטריצה של NxW וממלא אותה תא אחר תא. לא ניתן ליצור מטריצה של NxW תחת מספרים לא שלמים.

נשפר את האלגוריתם הנאיבי שראינו מקודם:

```
initialize arr[n][W] = undefined        -   נגדיר את מטריצת העזר לשמירת הכפילויות
def KS(n, W):
  if arr[n][W] != undefined:            -   נבדוק האם כבר חישבנו את הקומבינציה הזאת בעבר
      return arr[n][W]                  -   נחזיר את הערך השמור
  if n == 0 or W == 0:                  -   אם עברנו על כל הפריטים אם שאין יותר מקום בתיק
      result = 0                        -   מסתיימת הרקורסיה
  else if w[n] > W:                     -   אם משקל הפריט גדול מהמשקל שנותר בתיק
      result = KS(n-1, W)               -   נדלג על הפריט הנוכחי
  else:
      tmp1 = KS(n-1, W)                 -   חישוב האפשרות בלי לקיחת את הפריט הנוכחי
      tmp2 = v[n] + KS(n-1, W - w[n])   -   חישוב האפשרות עם לקיחת את הפריט הנוכחי
      Result = max { tmp1, tmp2 }       -   נבחר את המקסימום בין המקרים
  arr[n][W] = result                    -   נשמור את הערך במטריצת הכפילויות
  return result                         -   התוצאה הסופית (הערך המקסימלי)
```

מספר הקומבינציות הייחודיות (unique) האפשריות לקריאת הפונקציה הם: $N * W$.

וכך, ע"י שמירת הכפילויות, אלגוריתם זה מגיע לסיבוכיות זמן ריצה של $O(N * W)$.

האלגוריתם הגרידי (החמדני):

אלגוריתם זה מדרג כל פריט ע"י חישוב היחס בין ערך הפריט למשקלו (ratio), ממיין את כל הפריטים בסדר יורד, ומתחיל לאסוף את כל הפריטים שניתן להכניס לתיק לפי סדר דירוגם.

אלגוריתם זה הוא אלגוריתם קירוב בלבד ולא תמיד ייתן את התשובה האופטימלית (בגירסת 0-1 של הבעיה).

לדוגמא:

גודל התיק: 8 ק"ג

פריט 1: משקל - 5 ק"ג, ערך - 15

פריט 2: משקל - 4 ק"ג, ערך - 10

פריט 3: משקל - 4 ק"ג, ערך - 10

לפי הגדרת האלגוריתם, נחשב את כל דירוגים של כל הפריטים:

פריט 1: 15/5 ===> 3

פריט 2: 10/4 ===> 2.5

פריט 3: 10/4 ===> 2.5

הפריט הראשון שהאלגוריתם יקח הוא פריט 1, כי הוא בעל הדירוג המרבי, ואז אין יותר מקום לעוד פריט ואז הערך המקסימלי שהאלגוריתם ייתן הוא 15.

לעומת זאת, הפתרון האופטימלי הוא לקחת את הפריטים 2 ו-3 ובכך להגיע ל8 ק"ג וערך מקסימלי של 20.

הגם שאלגוריתם זה אינו אופטימלי, הוכח שהוא מבטיח פתרון קירוב של לפחות מחצית מהפתרון האופטימלי.

לאלגוריתם זה סיבוכיות זמן ריצה של $O(NLogN)$.

האלגוריתם הגנטי:

אלגוריתמים גנטיים (אבולוציונים) מתארים משפחה של אלגוריתמים המשמשים בין היתר לבעיות אופטימיזציה, כאלו שלא ידוע עבורן פתרון מדויק שעובד בזמן סביר.
הרעיון מאחורי האלגוריתם הוא לשלב זה בזה אלמנטים של פתרונות אפשריים, ולהפעיל "ברירה מלאכותית" בהחלטה מי הם המועמדים לעבור לשלבים הבאים.

להלן נתאר את שלבי האלגוריתם:

שלב 1 - ההתחלה (Start):

בשלב זה יוצרים את האוכלוסיה הראשונה שהיא למעשה הדור הראשון של התהליך הגנטי.

לרוב שלב זה נעשה באופן רנדומי לחלוטין, ולכן, גם אנחנו בחרנו להגריל מספר פריטים באופן אקראי לכל ישות באוכלוסיה.

כל ישות אצלנו מיוצגת כמערך בגודל כמות הפריטים כך שלכל פריט ישנו אינדקס, וערך התא במערך באינדקס זה הוא בינארי (0 או 1) כך ש0 מייצג - הפריט לא נלקח, ו-1 מייצג - הפריט נלקח.

לדוגמא:

1	0	0	1

עבור ישות זו הפריטים 1 ו4 נלקחו (2 ו3 לא נלקחו)

שלב 2 - התאמה (Fitness):

בשלב זה מדרגים כל ישות באוכלוסיה עד כמה היא מתאימה או קרובה לפתרון.
בבעיה שלנו הגדרנו פונקצית התאמה המחזירה את סכום ערכי הפריטים בישות, או 0 במידה ומשקלם הכולל עבר את המשקל המקסימלי של התיק.

* לאחר כמה נסיונות ראינו שאם נעלה בריבוע את הפונקציה שהגדרנו נבליט יותר את הפערים בין ישויות דומות - וכך ייעלנו את האלגוריתם עוד יותר.

שלב 3 - יצירת הדור הבא (New Generation/Population):

יוצרים את "ילדי" האוכלוסייה החדשה כגודל האוכלוסייה הקיימת כך שיצירת ילד תתבצע באופן הבא:

שלב 3.1 - בחירת הורים (Selection):

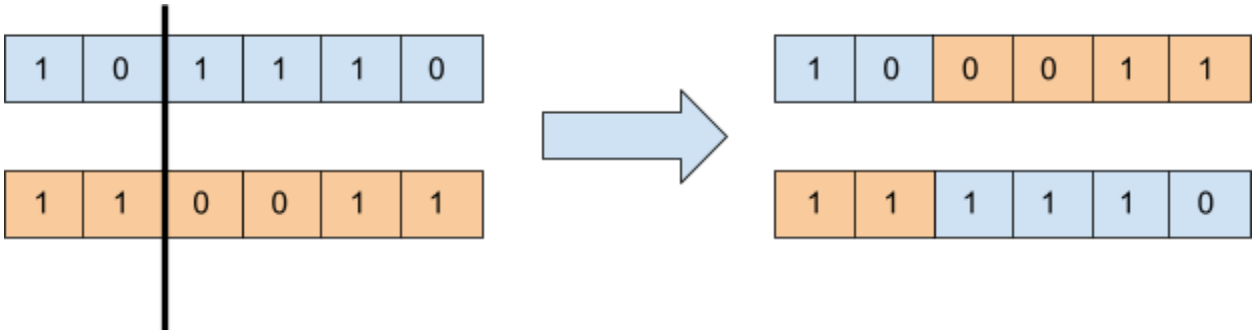
בכדי להגדיל את הסיכויים להגיע לילד "מוצלח" יותר (בעל התאמה גבוהה יותר), נרצה "לזווג" בסבירות גבוהה יותר בין ישויות שערכי ההתאמה שלהם גבוהים. מצד שני, אנו לא רוצים להתעלם לגמרי מישויות שערכם הוא אינו הגבוה ביותר שכן ייתכן שיש בהם מידע שהוא חשוב לנו. לכן, נרצה ליצור איזשהו מנגנון שנותן יותר משקל לישויות שערך התאמתם גבוה יותר. במקרה שלנו אנו בחרנו להשתמש במעין "גלגל רולטה" כך שלכל ישות יש כמות מקומות ברולטה כמספר ערך התאמתה (fitness).

(לדוגמא: לפריט עם ערך כולל של 10 יהיו 10 מקומות ברולטה, ולפריט עם ערך כולל של 5 יהיו רק 5)

שלב 3.2 - הזיווג (Crossover):

לאחר שהגרלנו שני הורים ברולטה שיצרנו, נרצה ליצור מהם ילד חדש. ישנם כמה דרכים לעשות זאת, אנו בחרנו להגריל אינדקס במערך ולקחת את החלק השמאלי מהורה אחד ואת הימני מהשני.

המחשה:



שלב 3.3 - השתקה (Mutation):

לאחר שיצרנו את הילד נרצה להכניס איזשהו מנגנון רנדומי על מנת לייצר עוד אופציות שאולי לא היו בדורות הקודמים. הדרך המקובלת לעשות זאת היא לעבור על כל האינדקסים במערך הישות החדשה ובסיכוי מסויים (מאוד נמוך) - להחליף את הערך בתא מ-0 ל-1 או להיפך.

הסיכוי אמור להיות נמוך כדי לא להפוך את האלגוריתם לרנדומי לחלוטין אבל מצד שני צריך גם לתת איזושהי תרומה שתהיה בעלת השפעה כלשהי באלגוריתם.

הצורה המקובלת היא השתקה בסיכוי של 1%, וכך מומש גם אצלנו.

שלב 4 - היתרבות (Reproduction):

בשלב זה מחליטים אם רוצים להפסיק את תהליך ההתרבות או להמשיך בתקווה לקבל התאמה יותר טובה מאחד הישויות. במקרה שנרצה להמשיך את ההיתרבות נחזור שוב ל**שלב 2**.

לאלגוריתם זה סיבוכיות זמן ריצה של $O(P * G * N)$.

(כאשר P - כמות הישויות באוכלוסיה, G - מספר הדורות)

חשוב לשים לב (על האלגוריתם הגנטי):

- ככל שיהיו יותר ישויות באוכלוסיה נקבל יותר ישויות מגוונות אך מצד שני יאטו את ריצת האלגוריתם.
- ככל שנבצע יותר היתרבויות כנראה נוכל להיחשף לפתרונות טובים יותר, אך שוב, נגדיל את זמן הריצה.
- ככל שערך הmutation (השתקה) גדל, נוכל להיחשף ליותר ישויות שלא בהכרח היו ביצירת האוכלוסיות, אך מצד שני, נגדיל את רנדומיות האלגוריתם.

הניסויים:

בדקנו את האלגוריתם הנאיבי (ממש מעט רק כדי להבין שהוא מאוד יקר) ואת שני אלגוריתמי הקירוב (הגרידי והגנטי) בהשוואה לאלגוריתם התכנון הדינמי בכדי לקבל אינדיקציה כלפי התוצאה האופטימלית.

לכן, בניסוי השתמשנו במספרים שלמים (למשקלי התיק והפריטים וכן לערכיהם) על אף ששלושת האלגוריתמים הנבחנים יכולים להתמודד גם עם מספרים שאינם שלמים.

בכל הניסויים השתמשנו בנתונים הבאים:

- גודל התיק: 1,000
- כמות הפריטים: 100
- כמות תרחישים: 10,000
- השתקה: 1%
- לאלגוריתם הגנטי:
  - מספר הישויות באוכלוסיה: 100
  - מספר הדורות: 500

* אין נוסחה מדויקת שלפיה נקבעים הפרמטרים, לכן מספר הישויות והדורות נבחרו למקרה זה אחר הרצות על ידי ניסוי וטעייה עד שהגענו לתוצאות טובות ומספקות ע"י שמירה על זמן ריצה יחסית נמוך עם תוצאות גבוהות ככל הניתן. (יתכן שלנתוני ניסוי אחרי תהיה קונפיגורציה אחרת מתאימה יותר).

* לאלגוריתם הנאיבי לוקח כ3 שניות להריץ תרחיש עם 30 פריטים, מחישוב סתמי נגיע לכך שעבור 100 פריטים (בהינתן שהאלגוריתם הינו בעל סיבוכיות אקספוננציאלית) נצטרך להריץ תרחיש אחד כמתואר בניסוי במשך $3 * 2^{70}$ שניות שזה כמה טריליוני שנים שאין לנו זמן לחכות...

לכן נבחן בקפידה רק את אלגוריתמי הקירוב בהשוואה לאלגוריתם התכנון הדינמי.

הנתונים:

ניסוי מספר 1:

הפריטים הוגרלו בצורה הבאה:

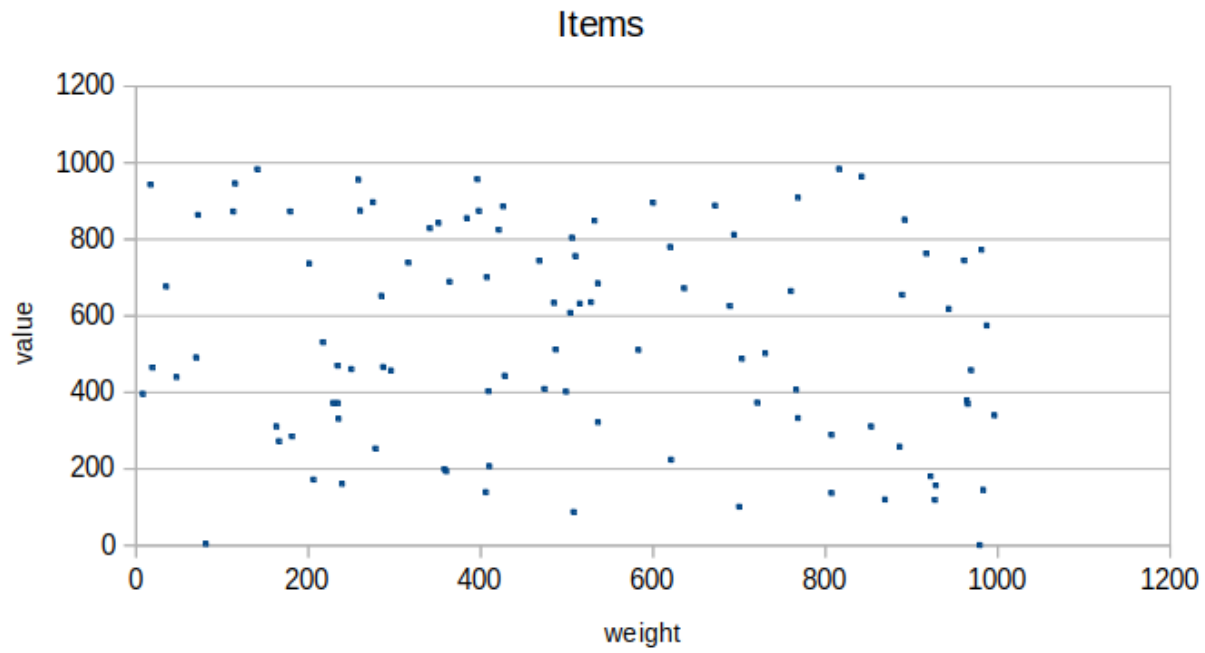
כל פריט יכול היה לקבל ערך בין 1 ל-1000 וכן משקל בין 1 ל-1000 באופן בלתי תלוי לחלוטין.

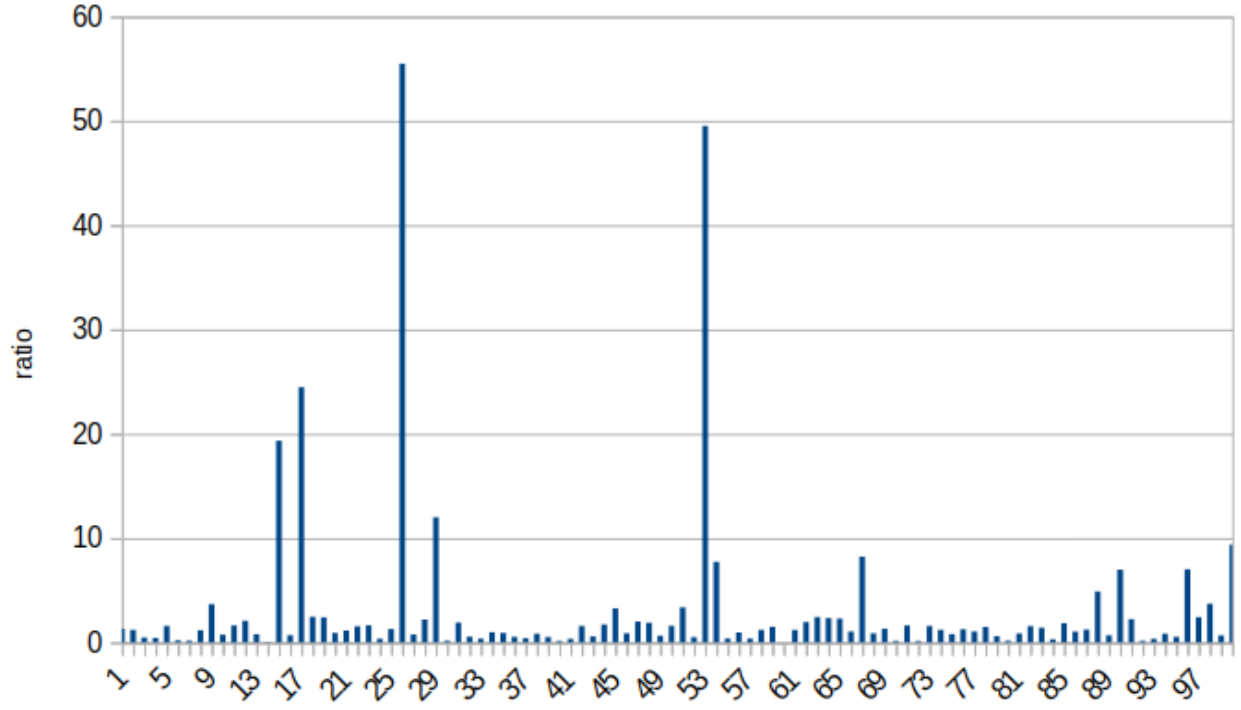
* ערכי הratio האפשריים הם בטווח 0.01-1000.

תוצאות הניסוי:

	זמן ריצה (ms) (ממוצע)	אחוז דיוק ביחס לפתרון האופטימלי (ממוצע)
האלגוריתם החמדני	<1	96.77
האלגוריתם הגנטי	90	95.32

תרחיש לדוגמא:





ניסוי מספר 2:

הפריטים הוגרלו בצורה הבאה:

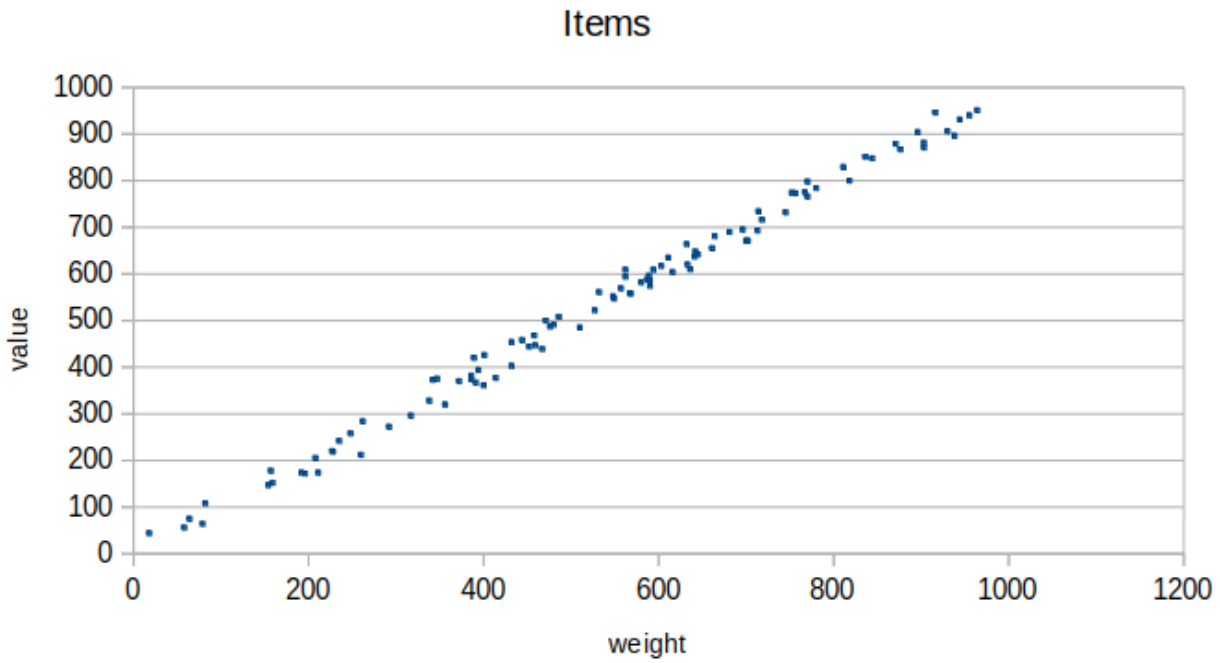
כל פריט יכול היה לקבל ערך בין 1 ל-1000 ומשקל בין 1 ל-1000 כך שההפרש בין הערך למשקל לא עולה על 50. (לדוגמא: עבור פריט עם ערך של 100 הערכים האפשריים למשקל הם 50-150)

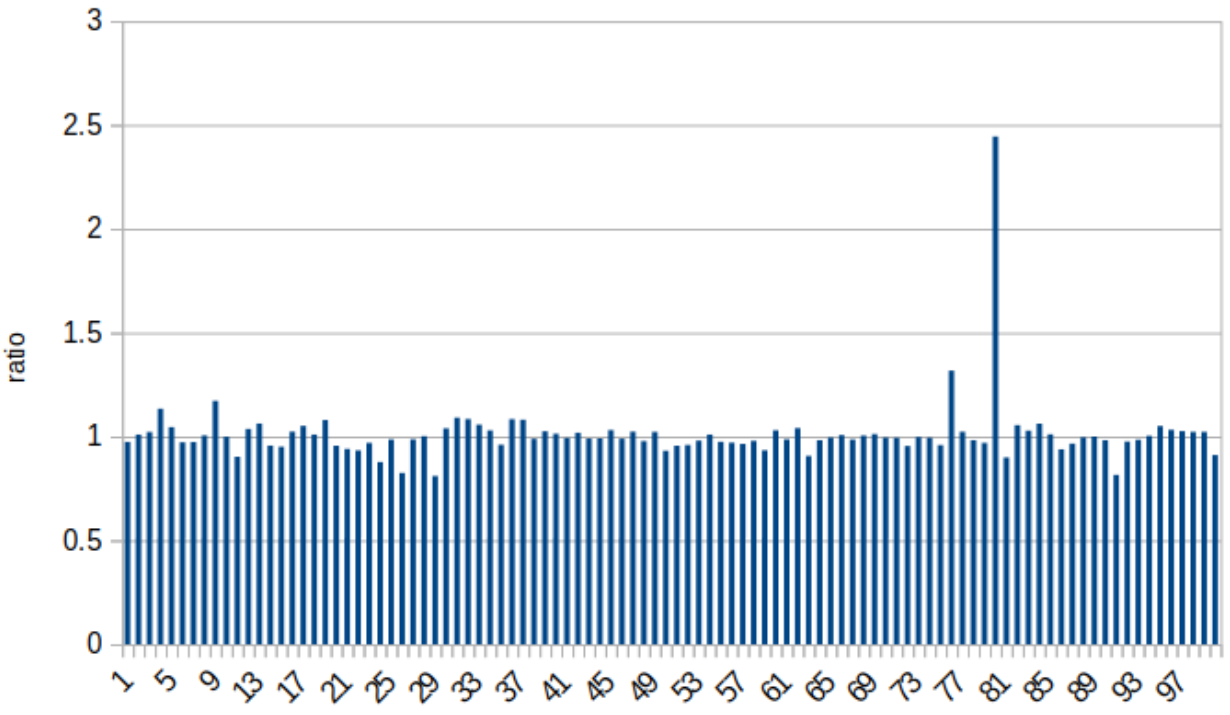
* ערכי הratio האפשריים הם בטווח 0.02-50.

תוצאות הניסוי:

	זמן ריצה (ms) (ממוצע)	אחוז דיוק ביחס לפתרון האופטימלי (ממוצע)
האלגוריתם החמדני	<1	87.96
האלגוריתם הגנטי	106	96.17

תרחיש לדוגמא:





ניסוי מספר 3:

הפריטים הוגרלו בצורה הבאה:

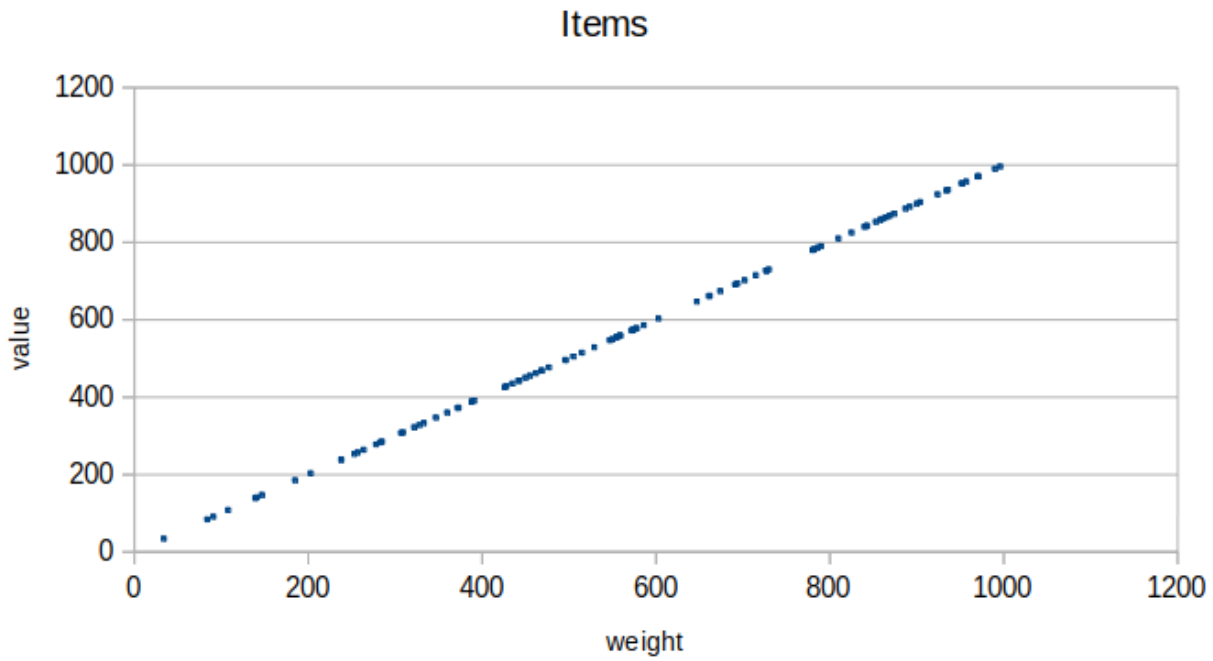
כל פריט יכול היה לקבל ערך ומשקל זהים בין 1 ל-100.

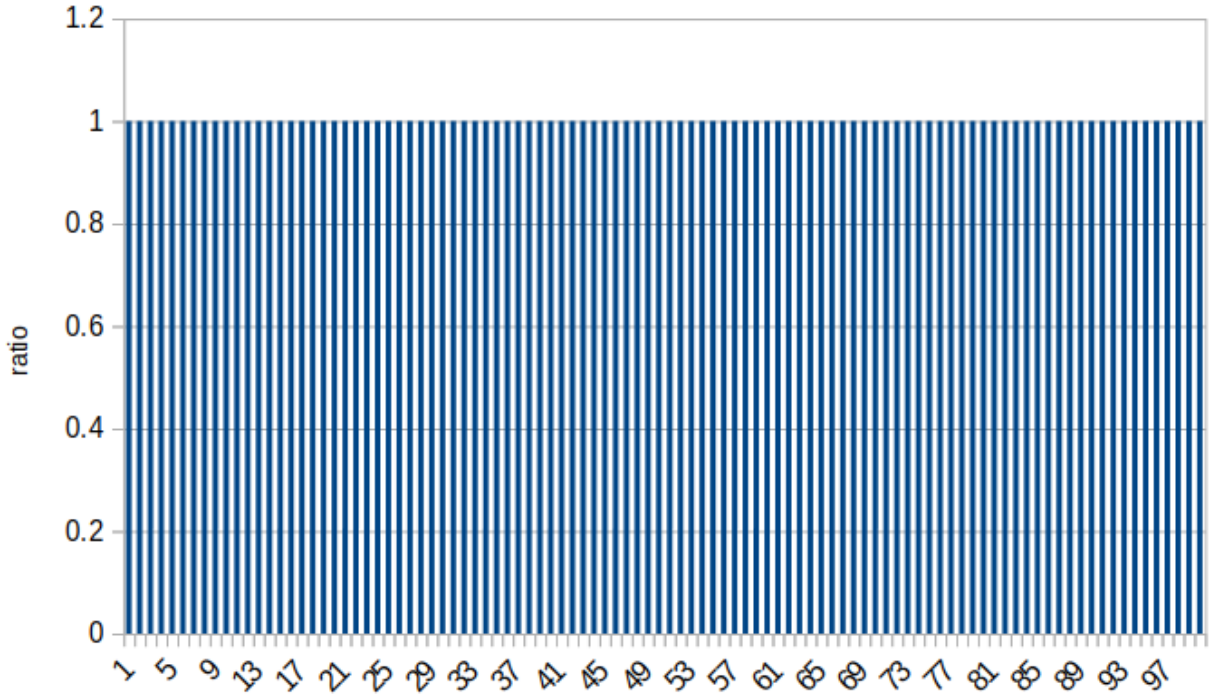
* ערכי הratio האפשריים הם 1 בלבד.

תוצאות הניסוי:

	זמן ריצה (ms) (ממוצע)	אחוז דיוק ביחס לפתרון האופטימלי (ממוצע)
האלגוריתם החמדני	<1	71.29
האלגוריתם הגנטי	114	100

תרחיש לדוגמא:





ניתוח התוצאות:

- מבין האלגוריתמים שהצגנו, המדוייק ביותר הינו האלגוריתם הנאיבי שכן הוא מכסה את כל הקומבינציות האפשריות ואכן ייתן לנו תשובה מדוייקת, אך הוא גם היקר ביותר (משמעותית) משאר האלגוריתמים.

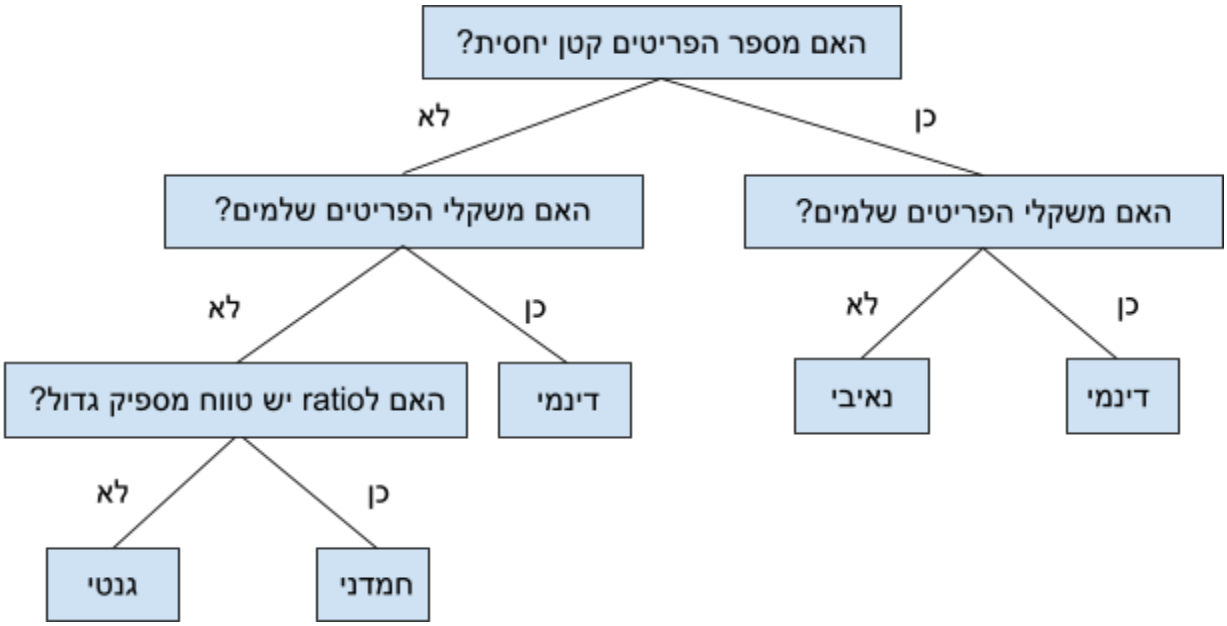
- אלגוריתם התכנון הדינאמי, הנותן גם הוא את התוצאה האופטימלית, ניתן לשימוש רק כאשר הבעיה מוגדרת תחת המספרים השלמים והוא גם המהיר ביותר בפתרון בעיה זו.

- האלגוריתם החמדני הינו מהיר יותר מהאלגוריתם הגנטי אך לעיתים מדייק פחות ממנו.
  - אם נסתכל על ניסוי 1 נראה ביצועים יחסית זהים בין שתי האלגוריתמים.
  - בניסוי 2 וניסוי 3 (שהוא יותר קיצוני לניסוי 2) נוכל לראות איך היעילות של האלגוריתם החמדני יורדת לעומת הגנטי. ההסבר לכך הינו מכיון שהאלגוריתם החמדני מדרג לפי הratio וכך בוחר את מי יעדיף להכניס לתיק. לכן, ככל שטווח הערכים האפשריים לratio יותר קטן כך גם גדל הסיכוי שיעדיף פריט אחד על פני פריט אחר אפילו שהוא משתלם במעט מהפריט השני ואולי עדיף לדלג עליו ולקחת במקומו 2 פריטים אחרים (או יותר). מניסוי 2 ו-3 ניתן לראות שככל שהפריטים מתקרבים לישר מהצורה ($y = x$ בפרט ו)לישר $y = mx$ בכלל - אנחנו מבטלים את היתרון של דירוג הפריטים והדיוק יורד. אך ככל שהפריטים מפוזרים בצורה אקראית במערכת הצירים (כמובן ברביע הראשון) - גדל הדיוק של האלגוריתם החמדני.

מסקנות והמלצות:

בהנחה שאנחנו מוכנים לחכות פרק זמן מסוים כלשהו (במקסימום), נגדיר שזהו הזמן שיקח לאלגוריתם הנאיבי לרוץ עם "מספר פריטים קטן יחסית".

המלצות אפשריות על פי עץ החלטה:



קישור לקוד הניסויים: