העבודה הבאה תעסוק במציאת אלגוריתם לבעיית האופטימזציה של בחירת מספר שחקנים קבוע בהתאם לאילוצי תקציב. בעיית הבחירה הנ"ל מופיעה במגוון תחומים והיא שייכת לתחום של אופטימזציה דיסקרטית ובפרט קומבינטורית. ראשית ננסח את פונקציית המטרה, לשם כך נניח שהקבוצה האופטימלית תהא מורכבת מקבוצת שחקנים שדירוגם האישי הוא המקסימלי ביותר ומספר השחקנים הכולל לא יעלה על 8 שחקנים ויהיה מורכב מ-2 רכזים C,3 סמול/ פאוור פורווארד SF/PF ו-3 שוטינג/פאוינט גארד PG/SG **.
ננסח את הבעיה הכללית באופן הבא

$$Max: \ \sum v_i x_i$$

$$subject\ to: \ \sum s_i x_i \le budget$$

$$\sum R_{ji} x_i \ \le required\ players\ in\ position\ j$$

$$x_i \in \{0,1\} \ R_i \in \{0,1\} \ \forall i = 1, 2, 3 .... n,$$

$v_i = player\ i\ evaluation,\ s_i = player\ i\ salary$

$x = 0\ if\ player\ i\ \ is\ not\ chosen,\ otherwise\ 1$

$R = 1\ ,\ if\ player\ i\ playes\ the\ required\ position,\ \ otherwise\ R = 0$

$position\ j \in \{1 = C, 2 = PG, 3 = SG, 4 = SF, 5 = PF\}$
$budget = 7M\$\ for\ this\ paper,\ in\ the\ code\ will\ be\ chosen\ by\ the\ user$
$required\ players\ in\ position\ C = 2,\ in\ position\ PG\ or\ SG = 3, in\ position\ SF\ or\ PF\ = 3$
$n,\ in\ this\ paper\ will\ represent\ amount\ of\ players\ in\ the\ data\ base,\ in\ actual\ wil\ be\ 350+$

$S = \{0,1\}^n$ תהווה קבוצה אפשרית או קומבינציה של שחקנים שנבחרו באופן כללי או לחילופין נאמר ווקטור $x \in R^n$ יהיה ווקטור פתרונות $1, represent\ chosen\ players$ $x = (1, 0, 0 \cdots 1 \cdots 0)$.
x יקרא פיתרון פיזיבלי לבעיית האופטימזציה והוא ייצג את קבוצת השחקנים שנבחרו/לא נבחרו מכלל המועמדים כל עוד הוא מקיים את האילוצים,קרי $||x|| < 8$ (הנורמה הסטנדרטית) ומקיים $s_i x \ \le budget = 13M\$$.
פונקציות המטרה פשוטה קמורה.אך אוסף הפיתרונות הפיזיבלי כמו רוב הבעיות הדיסקרטיות איננו קמור.
הנתונים ובנית המתודה לפיתרון הבעיה- הנתונים נלקחו ממאגר נתוני השחקנים של ליגת ה-NBA 20-21, לצורך דירוג השחקנים בחרנו את את הנוסחה EEF, נציין כי הנוסחה הינה בסיסית לשם מדד יעילות שחקן ואינה מדוייקת בהתאם לעמדה לזמן המשחק או השוואתית וכיוון שכן ראינו קורלציה בין הנוסחה לבין נתוני השכר נסתפק בנוסח הנ"ל וזאת במילא ניתנת לשיפור.

בשל הדימיון הרב לבעיה באופטימזציה קומבינטורית ידועה, בעיית תרמיל הגב (*knapsack problem*) בשלמים,שעבורה קיימים אלגוריתמים פולינימיאליים הרעיון היה להיעזר בשיטות דומות לפיתרון ולשם כך נחלק את הבעיה ל-3 תתי בעיות וננסח את פונקציה המטרה באופן הבא, פיתרון שלה יביא לפיתרון הכולל *ומעתה אתייחס בעבודה לפיתרון עבורה.*

$$Max: \ \sum v_i x_i$$

$$subject\ to: \ \sum s_i x_i \le budget\ for\ specific\ pos\ j \ \ ***$$

$$\sum x_i \ \le required\ players\ in\ position\ j \qquad\qquad j = 1 = C, 2 = SF/PF, 3 = SG/FG$$

**הרעיון לחלוקה של 3 תתי קבוצות ולא 5 קב' נפרדות (בהתאם למס' העמדות במשחק) היא לצורך התאמה למספר השחקנים הכללי המבוקש,8, מתוך הנחה שהשחקנים שנמצאים יחד באותה קטגוריה יכולים למלא את אותם תפקידים. בהנחה שכל דרישה אחרת לא תשפיע על האלגוריתם הכללי ותבוצע באופן דומה תוך כדי שינויים קלים בלבד.
*** החלוקה של התקציב לתת הקב' תהיה לבחירת המשתמש, לשם הרצה עבור C , $j = 1$ התקציב יהיה $3M$ עבור הקבוצה השניה SF/PF , $j = 2$ התקציב יהיה $5M$, ועבור הקב' האחרונה $j = 3$ SG/PG התקציב יהיה $5M$.

0-1 knapsack problem

בהינתן n פריטים עם משקל w_i ועם ערך v_i משוייכים לכל פריט, ותיק עם משקל W נרצה לדעת מהי הקומבינציה של פריטים שעלינו לבחור כך

שערכם הכולל יהיה המקסימלי ביותר Max: $\sum v_i x_i$ ולא יעלה על תכולת התיק W $.\sum v_i x < W_i$ $\quad w_i, p_i \in Z^+$

הפונקציה הרקורסיבית לפיתרון

$knapsackFunc(v, w, W, N)\{$
$return\ 0;$ $\quad if \quad W = 0\ or\ N = 0$
$knapsackFunc(v, w, W, N - 1)$ $\quad if \quad w_i > W$
$max\,\{v[i] + knapsackFunc(v, w, W - w[i], N - 1),$ $\quad else$
$knapsackFunc(v, w, W, N - 1)\}$

הערך המוחזר יהיה הערך האופטימלי של הקב' האופטימלית.

בשיטת התכנון הדינמי נחסוך את הקריאה הרקורסיבית על ערכים שכבר חושבו ע"י שמירת נתונים שחושבו במטריצת $M[m][n]$
m-כמספר המשקלים עד לW, n-מספר הפריטים.
לצורך מילוי הטבלה אנחנו בוחנים את אותם המקרים שנבחנו בניסוח הרקורסיבי של הבעיה
הרעיון הכללי הוא לבחון מקרים שונים בהם לתיק יש משקל w_i במקום W ולבחון האם לכל מקרה כזה פריט i יכול להיכנס, כלומר משקלו לא חורג במידה ולא נבחן האם המקסימום מתקבל איתו או בלעדיו מבין i הפריטים שנבדקו.

האלגוריתם:

1. אם $w_i > W$ לא ניתן להשתמש בפריט i והערך במטריצה יהיה זהה לקודם לו (שורה מעליו)= $M[i-1][j]M[i][j]$
2. אם $w_i < W$ ניתן להשתמש בפריט i, נבדוק מהו הערך המירבי איתו ובלעדיו
$M[i][j] = max\{M[i-1][j]\,, M[i-1][j - w[i]] + v[i]\ \}$
3. תנאי הבסיס אם המשקל או הפריט שווה ל-0 $M[i][j] = 0$,

נמלא את הטבלה עד לסופה הערך האחרון יציין את הערך האופטימלי שיוכל להתקבל.

פסאודו קוד-

i	v	w
1	5	4
2	4	3
3	3	2
4	2	1

Capacity=6

i \ w	0	1	2	3	4	5	6
0							
1							
2							
3							
4							

$M[i][j];$
$M[i][j] = 0, \quad if\ i = 0\ or\ j = 0$
$M[i][j] = M[i-1][j], \quad if\ \ w[i] > j$
$M[i][j] = max\{M[i-1][j]\,, M[i-1][j - w[i]] + v[i]\ \} \quad if\ w[i] < j$

A demonstration of the dynamic programming approach from [wiki]

שיחזור הפריטים שנבחרו מהמטריצה-

אלגוריתם:

נלך לשורה ה i עמודה ה-j נבדוק אם הערך שם שונה מהערך בשורה הקודמת לה

- *אם הוא שווה לה, הפריט ה-i לא השפיע ולכן לא נכלול אותו כלומר נסמנו 0*
- *אם הוא שונה,נכלול אותו בקבוצת פריטים שנבחרו, נסמנו ב-1.*

פסאודו-קוד

$while(j != 0)\{$
$if\ M[i][j] != M[i-1][j]\{$
$append\ item[i]\ to\ the\ set\ of\ chosen\ players$
$j = j - w[i]$
$\}$

בפסאודו קוד הפיתרון היא רשימת פריטים נבחרים ולא ווקטור פיתרון מלא.

ההבדלים בין שתי הבעיות (knapsack 0-1 לבעיית האופטימזציה שלנו) והתאמת האלגוריתם

- בבעיה שלנו היו מגבלות על מספר ה"פריטים" קרי שחקנים
- בבעיית ה-Knapsack המשקלים המקושרים לפריטים הם שלמים תנאי הכרחי לכך שהפיתרון יהיה בזמן פולינימיאלי (אחרת מספר עמודות המטריצה בתיכנון הדינמי יהיו גדולות מדי לדיוק עשרוני), אצלנו ה"משקלים" קרי השכר איננו מופיע בשלמים.

בעבור ההבדל האחרון גישרנו על הפער ע"י עיגול שכר שלכל שחקן כלפי מעלה לחלוקה כרצוננו בקוד בחרנו לעגל לשלמים וחצאים. שכר השחקנים מעוגל למספרים 0.5M, 1M,1.5M,,2M וכו' (ניתן לבצע חלוקה עדינה יותר אך כזאת שלא תגדיל את זמן הריצה לאין ערוך) ובכך הגדלנו את מספר העמודות אצלנו פי 2 כשכל עמודה מייצגת משקל של חצי מיליון.

בעבור ההבדל הראשון הוספת אילוץ על מספר השחקנים נוסיף מימד נוסף למטריצה שיגביל את מספר השחקנים, וזאת ידועה כגירסה מורחבת של בעיית תרמיל הגב $multi - dimensonal\ knapsack\ problem$ המטריצה שלנו תהיה מהצורה $M[n][m][k]$ המימד הנוסף k יהיה המגבלה על מספר השחקנים שנרצה לבחור, שאר המשתנים, n מספר השחקנים הכולל במאגר הנתונים,ו- m יהיה התקציב.

לשם כך היה עלינו לשנות מעט את האלגוריתם למילוי המטריצה על מנת להתחשב במימד הנוסף
ההבדל מהאלגוריתם הקודם של בעיית $knapsack$ הרגילה
אם $w_i < W$ ניתן להשתמש בפריט i, נבחר במקסימום מבין האפשרויות איתו ובלעדיו

$$max\{M[i-1][j][k]\ , M[i-1][j-w[i]][k-1] + v[i]\ \}$$

$$M[n][m][k]$$
$$\begin{array}{ll} 0, & if\ n = 0\ or\ m = 0,\ k = 0 \\ M[n-1][m][k], & if\ \ w[i] > j \\ max\{M[n-1][m][k]\ , M[n-1][m-w[i]][k-1] + v[i]\ \} & if\ w[i] < j \end{array}$$

אלגוריתם לשיחזור פריטים שנבחרו
בפועל למרות שהמטריצה היא תלת ממדית נוח לחשוב עליה כ-n מטריצות $M[j][k]$
השיחזור על $M[j][k]$ מתבצע באופן דומה אנחנו הולכים לאיבר האחרון ובודקים מתי הערך הנ"ל נראה לראשונה ע"י מעבר כל פעם $j - l$ (במעלה השורות של המטריצה $M[j][k]$) במידה וזוהה שינוי
אנחנו בודקים האם השחקן ה-i השפיע על שינוי הערך,ע"י בדיקה תמיד בנוסף בטבלה ה $n - i$ של $M[j][k]$
במידה ואנחו מזהים שינוי אנחנו עוברים ל- $M[j - player[i].salary][k-1]$ כלומר מורידים
את מספר האילוצים ומפחיתים את שכר השחקן שנבחר וממשיכים באופן דומה עד שאחד הערכים מתאפס.
.
במידה ושחקן השפיע על הערך בתא

$set\ of\ chosen\ players = []$
$players\ limit = l$
$while(set\ of\ chosen\ players < l\ and\ k > 0)\{$

$\quad while(dp[i][j][k]\ ==\ dp[i-1][j][k])\{$
$\quad\quad i\ -=\ 1\}$
$\quad chosen\_players.append(player\_list[i-1])$
$\quad while(dp[i][j][k]\ ==\ dp[i-1][j][k])\{$
$\quad\quad j\ -= 1\}$
$\quad j = j - w[i]$
$k = k - 1$
$i = i - 1\}$

סיבוכיות זמן $k << m$, $O(m * n * k)\ \sim O(n^2)$, יותר טוב מחיפוש ממצה של כל האפשרויות naive brute force שיקח $O(2^n)$

ההשוואה בין הקוד לחבילה קיימת pywraplp של Google OR TOOL בין היתר משתמשת באלגוריתם *branch and bound* משופר ומשולב בעוד כמה אלגוריתמים.

לשם השוואה נסקור בקצרה את דרך הפיתרון לבעיה ע"י האלגוריתם הנ"ל-

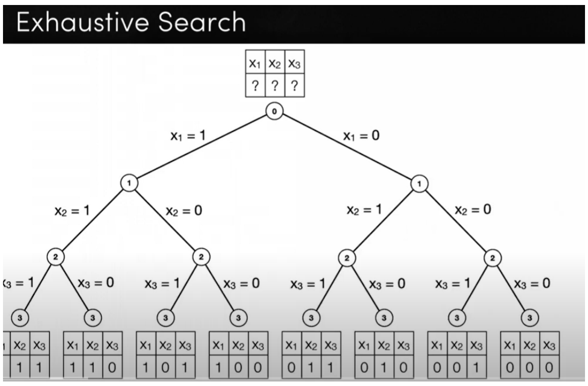
אלגוריתם branch and bound (סעף וחסום)-
הרעיון הכללי לקחת את עץ ההחלטה* לחיפוש ממצה של האפשרויות ולחסוך חיפושים מיותרים.
בונים עץ החלטה כל צומת תעדכן את הערכים הבאים value,room,estimate
value-ערך הקב' עד לאותה הצומת כולל
room-מס' שחקנים שנותר להוסיף
estimate-פונקצית שיערוך לערךהאופטימלי שיקבע בהתחלה ותעודכן בכל צומת בה אנחנו בוחרים לא לקחת שחקן, דוגמה להערכה גסה סכום ערכי v_i של כל שחקן.

2. Branching- כל רמה תבחן את צירוף השחקן ה-i הרמה הראשונה לדוגמה תבחן את צירופו של שחקן 1,כשענף אחד יבחן את האפשרות לצירוף השחקן וענף השני לא יכלול את השחקן כל צומת כזאת תעדכן את הערכים בהתאם לאפשרות שנבחנה.
3. Bounding- אם הגענו לענף מסויים שהגיע לאחת מהמגבלות ויש לנו פיתרון פיזבילי נוכל להשתמש בvalue שלו להשוואה אם לחקור ענפים אחרים, במידה ונראה שלא נגיע לפיתרון טוב יותר נפסיק את החיפוש בו. מקרה שני להפסקת חיפוש אם בהתקדמות נראה כי הפיתרון לא פיזבילי כמו כן נוכל להפסיק את החיפוש בו.

ישנן שיטות שונות לשיפור האלגוריתם נבחן מקרה בו אנחנו משפרים את הערכה הראשונית נוכל למצוא חסם תחתון טוב יותר לשיערוך האופטימלי שלנו ע"י relaxtion של אחד האילוצים, לדוגמה פה נהפוך את האילוץ $x \in \{0, 1\}$ללא פעיל ונמצא את הפיתרון החמדני האופטימלי לבעיה בהנחה ש-x יכול להיות שבר (מובטח פיתרון אופטימלי). על מנת לפתור זאת נסדר בסדר יורד את השחקנים הערך $value/salary$ ונבחר את כל השחקנים שנוכל לבחור עד שנהיה קרובים למלא את האילוץ של התקציב בהפרש שנותר נכניס רק חלק מהשחקן הבא שניתן להשלים ובהתאם לכך נוסיף גם את חלק מערכו לערך הכולל של שחקנים שנבחרו ונקבל חסם תחתון אופטימלי.

יתרונות השיטה- בניגוד לשיטה שבחרנו לא יהיה צריך לעשות התאמות לבעיה הראשית שלנו כמו להוסיף מימד נוסף או לחלק את פונקציית המטרה לתת פונקציות כל אילוץ נוסף יתן לנו אינדיקציה נוספת אם לחקור רמה נוספת, קרי צירוף שחקן נוסף.
אין צורך בהתאמת נתונים כמו עיגול שכר.
אין צורך בשיחזור שחקנים שנבחרו נתונים של כל ענף נשמרים תוך כדי ריצה, מבני נתונים יעיל יותר רשימה מקושרת ולא מטריצה.
חסרון- זמן ריצה במקרה הגרוע יכול להיות קרוב ל naive brute force $O(2^n)$

* חיפוש ממצה לדוגמה בהינתן 3 שחקנים 2^3 אפשרויות לבחון,
כל רמה בוחנת האם לצרף את שחקן x_i. בהתאם לנתונים המשוייכים לאותו שחקן וההאילוצים.



בקוד עצמו בוצעה השוואה של הבעיה על קבוצה קטנה של שחקנים התקבלו אותן תוצאות ולא נראה הבדל גדול בזמנים התיכנון הדינמי היה מעט יותר מהיר מהחבילה של google בממוצע אך יכולים להיות דברים נוספים שהשפיעו על זמן הריצה.

ישנן מגוון גישות ושיטות לפיתרון הבעיה שלא הובאו בעבודה כאלו המבטיחים פיתרון אופטימלי או קירוב לפיתרון האופטימלי כגון אלגוריתמים היוריסטיים חמדניים, גנטיים.