**חלק א' – The Multi Knapsack Problem:**

* **פירוט ייצוג הבעיה**

עבור הבעיה יצרנו את המחלקות הבאות, מימוש בקובץ Knapsack.py:

מחלקת Knapsack – המייצגת אובייקט של הבעיה:

* אובייקט ממחלקה זו מכיל את השדות הבאים:

m – סך האובייקטים שיש בבעיה.

n – סך השקים שיש בבעיה.

sacks – רשימת השקים של הבעיה, מכילה אובייקטים מסוג Sack, יפורט בהמשך.

items – רשימת הפריטים של הבעיה, מכילה אובייקטים מסוג Item, יפורט בהמשך.

items\_used – רשימת הפריטים אשר נמצאים כרגע בשקים, בסיום מכיל את הפריטים המייצגים את הפתרון שנמצא.

value – סכום הפריטים אשר נמצאים כרגע בשקים, בסיום הינו ערך הפתרון המתקבל מהחיפוש.

opt – הפתרון האופטימלי כפי שצוין בקובץ הבעיה.

* לאובייקט מסוג זה יש את הפונקציונאליות הבאה:

להוסיף / להוריד פריט מהשקים

בדיקת חוקיות השקים מבחינת capacity

מטודת הדפסה

מחלקת Sack – המייצגת אובייקט מסוג שק:

* אובייקט ממחלקה זו מכיל את השדות הבאים:

number – ספרור השק בבעיה.

capacity – הנפח של השק.

room – סך המקום שנותר להכניס לשק.

items – רשימת הפריטים אשר נמצאים בשק.

value – ערך השק עבור הפריטים שנבחרו.

* לאובייקט מסוג זה יש את הפונקציונאליות הבאה:

הוספה / הורדה של פריט מהשק

מטודת הדפסה

מחלקת Item – המייצגת אובייקט מסוג פריט:

* אובייקט ממחלקה זו מכיל את השדות הבאים:

number – ספרור הפריט בבעיה.

value – ערך הפריט.

weights – מערך המשקולות של הפריט לפי שק.

* **מימוש ופתרון הבעיה בעזרת היוריסטיקת branch and bound ו-LP Relaxation**

מימוש בקובץ LDS\_search.py.

תהליך החיפוש LDS, כפי שנלמד בהרצאה, מתבצע באמצעות חלוקה ל-waves, כאשר כל wave מאפשר מספר טעויות בהיוריסטיקה השווה למספר ה-wave. כלומר, עבור ה-wave הראשון ננסה להגיע לפתרון ללא טעויות בהיוריסטיקה, ב-wave הבא נאפשר לטעות פעם אחת וכן הלאה.

לשם מימוש שתי הרילקסציות יצרנו פונקציה שכל פעם תחפש עם waves עולים.

Relaxation I – שק לא חסום:

למימוש הרילקסציה הזאת, הנחנו כי השק אינו חסום, מה שמתאים להיוריסטיקה בה כל הפריטים נכנסים לשק.

על כן, הכנסנו את הפריטים בזה אחר זה, עד לקבלת שק לא חוקי.

במידה וה-wave בו אנו נמצאים ברקורסיה אינו 0, ננסה בכל פעם להוציא פריט אחד מהפריטים שהכנסנו, וננסה להמשיך בחיפוש מאותה נקודה עם wave אחד פחות.

לדוגמה:

אם בבעיה מסוימת הכנסנו את הפריטים 1 2 3, וכאשר ניסינו להכניס את הפריט 4, נתקלנו בשק החורג מנפחו, אזי, אם ה-wave אינו 0, ננסה להוציא את פריט 4, ולהמשיך לפריט עם wave-1.

נמשיך בתהליך זה, עד ש-wave הינו 0, ואז נעריך את הפתרון שהתקבל.

לאחר מכן, ננסה גם עבור מקרים בהם לא בחרנו את פריטים 3 2 1 (במקום פריט 4).

בכל שלב באלגוריתם, נוכל להעריך את המקסימום אליו ניתן להגיע במקרה הטוב ביותר באמצעות סך הפריטים שכבר בשק ועוד סך הפריטים שטרם השתמשנו אך לא ויתרנו עליהם.

במידה וחסם עליון זה הינו פחות טוב מהפתרון הטוב ביותר אשר מצאנו עד כה, נוכל לגזום את תת העץ הזה ולהמשיך לחיפוש הבא.

אלגוריתם זה, מסתמך על כך שהוא בודק את הפתרונות הטובים ביותר בהתחלה, בוחר הכי הרבה פריטים בצורה חמדנית, ובכל פעם מוריד את כמות הטעויות ועל כן מסתמך שאם הפתרון לבעיה בעל הרבה פריטים אזי פתרון זה ימצא במהירות.

חתימת הפונקציה:

def unlimited\_sack\_search(self, wave, end\_time, dont\_use=None, last\_error=-1)

מימוש בקובץ שורות 54-86.

Relaxation II – הזנחת האילוץ לפריטים שלמים:

למימוש רלקסציה זו, מימשנו את הפונקציה neglecting\_integrality\_constraints במחלקת Knapsack, שורות 39-79, אשר מזניחה את האילוץ של הפריטים להיות שלמים, ובכל פעם, בוחרת את הפריט שהיחס בין המשקל לשווי שלו הינו הגדול ביותר. מכיוון שלכל פריט יש משקל שונה בכל שק אזי נבדוק עבור כל שק בנפרד מהו הפריט הטוב ביותר ובכך נחליט בסך הכל את מי לבחור בכל איטרציה. במידה והפריט הטוב ביותר לא נכנס בצורה חוקית לשק, נכניס את החלק שכן נכנס לשק ובכך נקבל חסם עליון על הבעיה בהינתן הפריטים שנבחרו.

שיטת החיפוש במקרה זה נעשת בצורה דומה לשיטת החיפוש שהצגנו ברלקסציה הקודמת, כאשר ההבדל הינו בסדר בחירת הפריטים ובדרך בה אנו משיגים חסם עליון על הבעיה.

כפי שראינו בהרצאה, שיטה זו, מביאה לגיזום אגרסיבי יותר, בנוסף לכך, בכל איטרציה תבחר את הפריט הטוב ביותר באופן יחסי, על כן היינו מצפים ששיטה זו תביא לתוצאות טובות יותר בזמן קצר יותר.

מנגד, במידה וקיים מצב שבו, לדוגמה, הפריט בעל היחס הטוב ביותר לא נמצא בפתרון האופטימאלי, ייתכן וייקח לנו זמן רב יותר למצוא את הפתרון האופטימלי, שכן אנו נבדוק קונפיגורציות שפריט זה אינו נמצא בהם רק לאחר שסיימנו לבדוק את כל הקונפיגורציות בהם פריט זה כן נמצא בהם.

חתימת הפונקציה:

def fractional\_search(self, wave, end\_time, dont\_use=None)

מימוש בקובץ שורות 26-51.

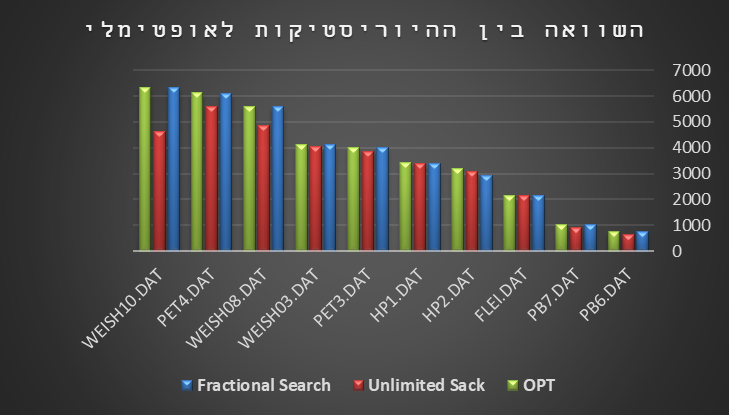
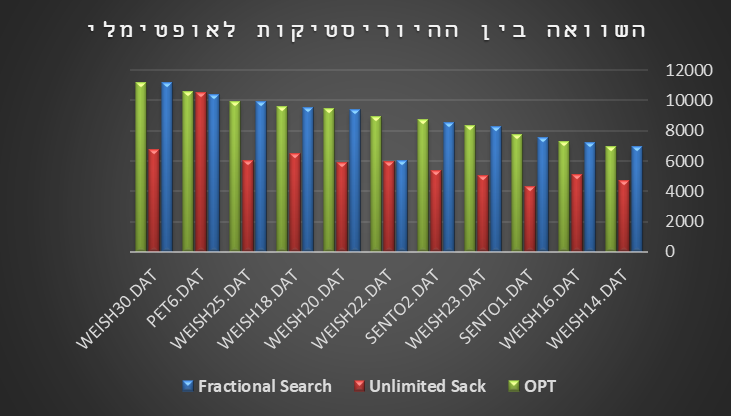
* **השוואת תוצאות ההרצה בין שתי שיטות הרלקסציה**

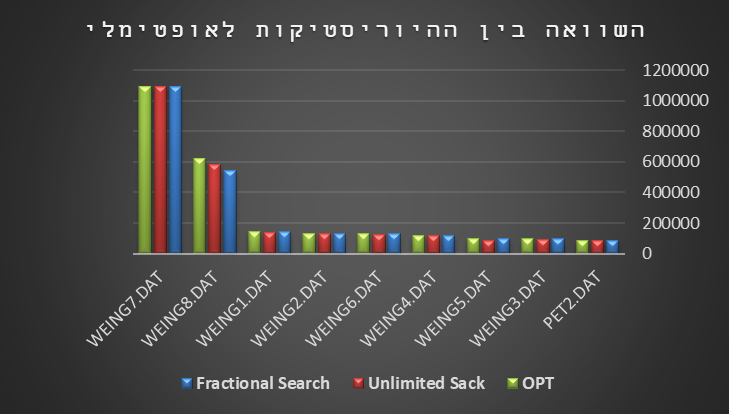
מכיוון שאלגוריתמים אלו הינם דטרמיניסטיים ושלמים, אזי מספיק הרצה אחת כבסיס השוואתי על מנת להעריך את הביצועים של האלגוריתמים. כלומר, ניתן להניח כי עבור שתי ריצות של אותו אלגוריתם על אותו קלט נקבל תוצאות דומות מאוד.

על כן, בחרנו להריץ על מספר רב של בעיות, סה"כ 30 בעיות, במקום להריץ פעמים רבות את אותה הבעיה.

לכל אלגוריתם ובעיה הקצבנו 120 שניות למציאת הפתרון הטוב ביותר שניתן להשיג בזמן הקצוב.

להלן תוצאות ההרצה



הסבר:

באופן כללי תוצאות הרלקסציה השנייה, Fractional Search, הניבה תוצאות טובות יותר בזמן הקצוב.

ניתן לראות זאת באופן מובהק בטבלה השמאלית העליונה, שבה הרלקסציה השנייה (עמודה כחולה) השיגה לרוב פתרון הקרוב מאוד לאופטימלי (עמודה ירוקה).

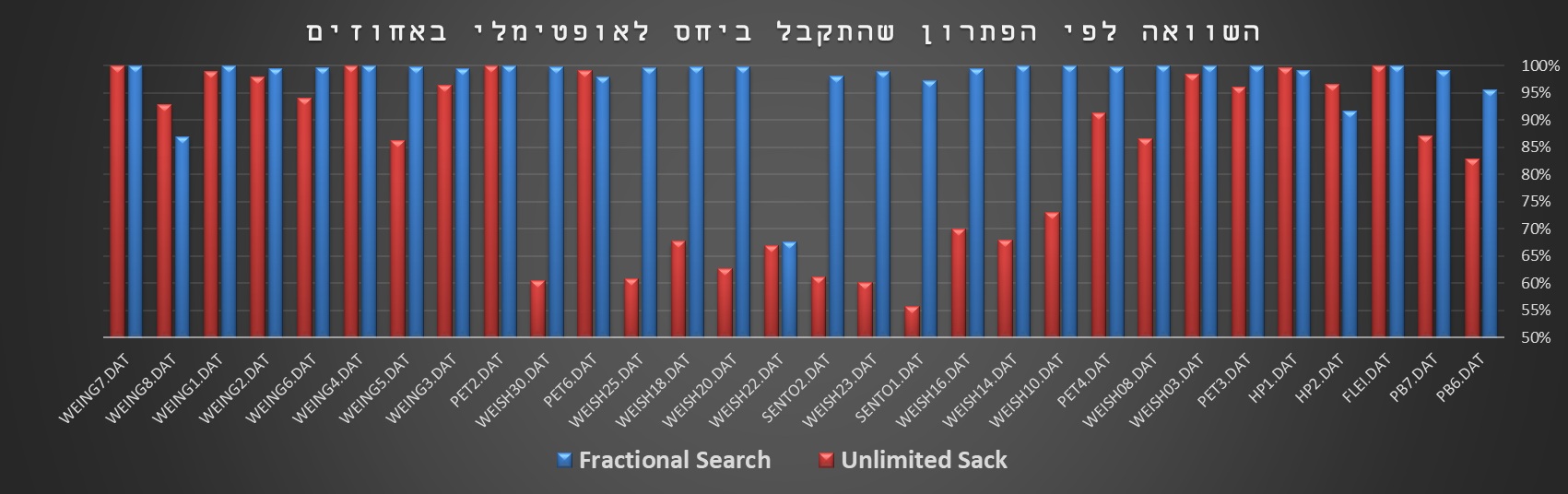
לעומת זאת, הרלקסציה הראשונה באופן יחסי רחוקה מהאופטימלי למעט מקרה אחד (בעיה PET6).

בשאר הגרפים, קיים יתרון פחות מובהק ל-fractional search, אך עדיין הינו עדיף ברוב המקרים.

השוואה על סמך אחוזי הצלחה ביחס לפתרון האופטימאלי

הוספנו טבלה אשר משקפת עד כמה כל אלגוריתם טוב באופן כללי ביחס לאופטימאלי ולא כהשוואה בין האלגוריתמים.

לשם כך, הצגנו את האחוז מתוך הפתרון האופטימלי אליו הגיע כל אלגוריתם.



הסבר:

על פי הטבלה שלעיל, הרלקסציה השנייה (עמודה כחולה), כמעט תמיד מגיע למעל 95% מהפתרון האופטימאלי, למעט שני מקרים, באחד מהם הגיעה לכ-67% מהפתרון ובשני הגיעה לכ-87% מהפתרון.

בנוסף, רלקסציה זו, מצאה את הפתרון האופטימאלי ב-9 מתוך 30 הבעיות.

מנגד, הרלקסציה הראשונה (עמודה אדומה), הניבה תוצאות פחות טובות באופן כללי, כאשר רק בשני מקרים הצליחה יותר מהשנייה, בבעיות WENIG8, HP2. בנוסף, ניתן לראות כי רק ב-11 מתוך 30 הבעיות הגיעה לתוצאות של יותר מ-95%, ובמקרים רבים הגיעה לתוצאות נמוכות יותר מ-70%.

רלקסציה זו, הגיעה לפתרון האופטימאלי רק ב-3 מקרים לעומת השנייה שהגיעה ב-9 מקרים.

* **סיכום חלק א'**

בסך הכל, ניתן לראות כי הרלקסציה השנייה, fractional search, הניבה תוצאות טובות יותר כמעט תמיד, כפי שסברנו טרם בדיקת התוצאות.

בנוסף, ברוב המקרים מגיעה לפתרון האופטימאלי בזמן הקצוב, 120 שניות, ועל כן עבור חלק ב' נבחר להשתמש בה.