וסים טנוס: 207866328
פארוק כריים: 314656869

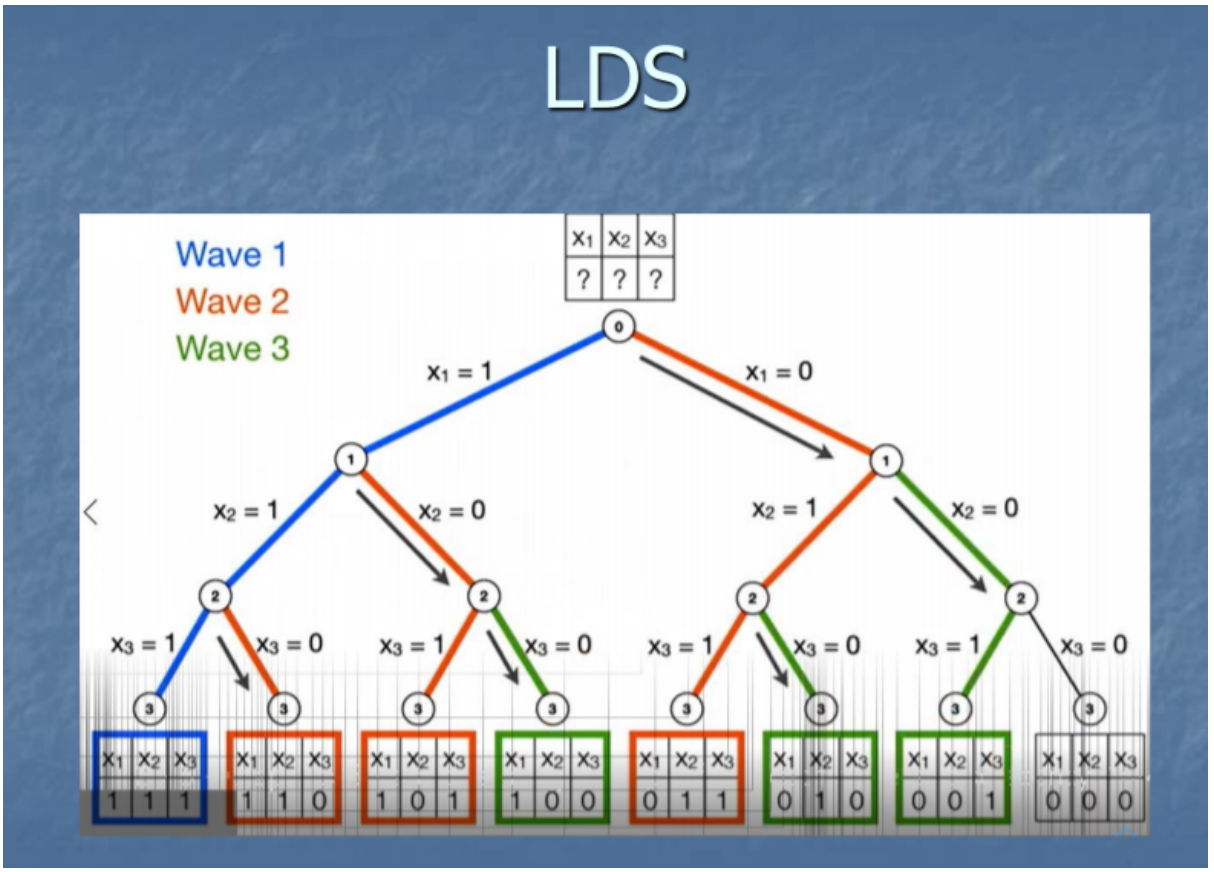
דוח מעבדה 4:

ניתן לערוך פרמטרים לאלגוריתמים בתיקיית INPUT. נא לוודא שה main באותה תיקיה ליד INPUT וARGS כדי שיראה אותם.

חלק א:

1. פתרו את הבעיות באמצעות חיפוש **LDS** והיוריסטיקת **branch and bound** ו- LP relaxation– חיפוש לעומק שאינו מפתח צמתים שערכם נמוך מהערך האופטימי המיטבי עד כה לפי ההיוריסטיקה ומבצע אופטימיזציה בין צריכת הזיכרון לבין יכולת הגיזום.

לפנינו תיאור האלגוריתם כמו שלמדנו בכיתה, שהוא אלגוריתם המחפש שכבות. (WAVES) - ש **LDS**



וסים טנוס: 207866328
פארוק כריים: 314656869

LDS:

לפי מה שראינו בכיתה, אנחנו מתחילים מוקטור שכולו 1-ים ואז מתחילים להחליף בתוכו (1-ים ל=> 0-ים) עם על הפרמוטציות.
פונקצית lds מתבלת בזה, והיא מקבלת :
את מספר האפסים .
הוקטור.
ומחזירה TRUE אם הגענו לאופטימום, FALSE אחרת.

```
def multiKnapsack(multiKnapsackProblem: MultiKnapsackProblem):
    global ARGS

    ARGS = multiKnapsackProblem
    length = multiKnapsackProblem.size
    print('optimum: ', multiKnapsackProblem.optimum)
    array = [i for i in range(length)]
    shuffle(array)
    for i in range(length):
        # for i in array:
        # for i in range(length, -1, -1):
        print('wave: ', i)
        lds(i)
        if ARGS.bestValue == ARGS.optimum:
            break

    print()
    print('best solution found:')
    print('vector: ', multiKnapsackProblem.bestVector, ', value: ', multiKnapsackProblem.bestValue)
    print('optimum: ', multiKnapsackProblem.optimum)
    if multiKnapsackProblem.bestValue == multiKnapsackProblem.optimum:
        print('reached optimum !!!!!!!!!')
```

וסים טנוס: 207866328
פארוק כריים: 314656869

זאת הפונקציה ש-מתבלת בכל הפרמוטציות עבור מספר אפסים קבוע.

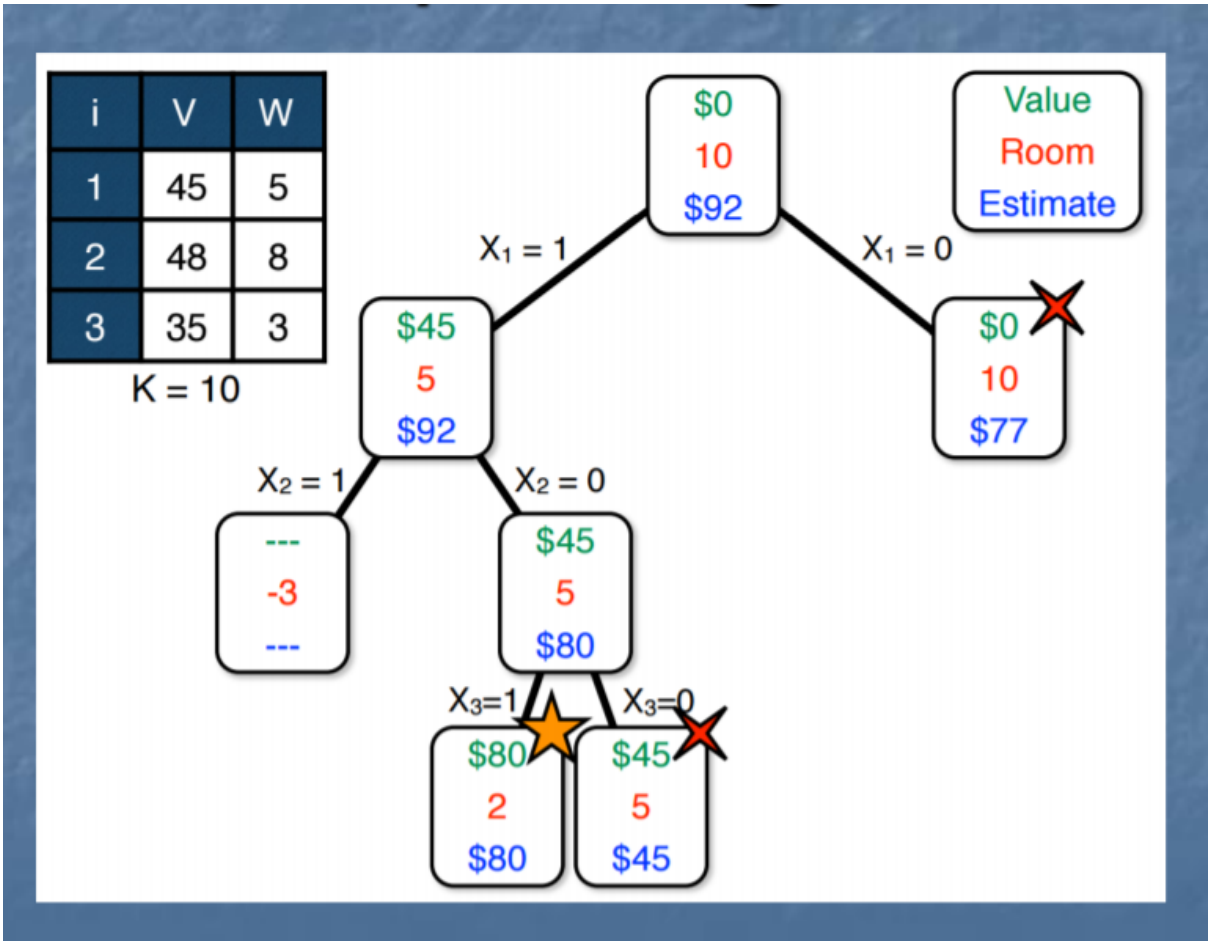
```
def lds(numberOfZeros: int, vector=None) -> bool:
    if ARGS.bestValue == ARGS.optimum:
        return True

    if vector is None:
        vector = []
    if ARGS.deadEnd.get(str(vector), False):
        return False

    estimate, weights = checkVector(vector)

    if not weights or estimate <= ARGS.bestValue:
        ARGS.deadEnd[str(vector)] = True
        return False

    if len(vector) == ARGS.size:
        value = ARGS.sacks[0].calculateValue(vector)
        if value > ARGS.bestValue:
            ARGS.bestValue = value
            ARGS.bestVector = vector
            print('vector: ', ARGS.bestVector, ', value: ', ARGS.bestValue)
            return True
        return False

    vector0 = vector[:]
    vector0.append(0)
    vector1 = vector[:]
    vector1.append(1)

    if len(vector) + numberOfZeros == ARGS.size and numberOfZeros > 0:
        return lds(numberOfZeros - 1, vector0)
    elif numberOfZeros > 0:
        retVal1 = lds(numberOfZeros, vector1)
        retVal0 = lds(numberOfZeros - 1, vector0)
        return retVal0 or retVal1
    else:
        return lds(numberOfZeros, vector1)
```

וסים טנוס: 207866328
פארוק כריים: 314656869

מתי הפונקציה מחזירה FALSE :
הפונקציה פועלת באופן הבא כמו שמתואר בתמונה שלקוחה מההרצאה:

- אם עבור אובייקט מסוים קיבלנו שאחד השקים קיבל ROOM קטן מאפס -> אז זה לא טוב.
- ואם ה ESTIMATE הנוכחי הוא כבר קטן מהטוב ביותר -> זה לא טוב.
- אחרת נעדכן את הטוב ביותר בהתאם.

** הערה: חשוב לדעת שכל וקטור מייצג עלה בעץ, ואנחנו לא מחכים עד שנעבור את כל המסלול ונגיע לעלה לגלות שהמסלול הזה לא טוב, אלה אנחנו בודקים את זה על הדרך (לפי התמונה ולפי ההרצאה).



וסים טנוס: 207866328
פארוק כריים: 314656869

מצורפת דוגמת הרצה:

```
vector:  [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1] , value:  3325
iteration:  4
wave done !!

vector:  [1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1] , value:  3485
vector:  [1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1] , value:  3685
iteration:  5
wave done !!

vector:  [1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1] , value:  3825
vector:  [1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1] , value:  3845
vector:  [1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1] , value:  3965
vector:  [1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1] , value:  4005
iteration:  6
wave done !!

vector:  [1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1] , value:  4015
iteration:  7
wave done !!

iteration:  8
wave done !!

iteration:  9
wave done !!

iteration:  10
wave done !!

iteration:  11
wave done !!

iteration:  12
wave done !!

iteration:  13
wave done !!

iteration:  14
wave done !!


best solution found:
vector:  [1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1] , value:  4015
optimum:  4015
reached optimum !!!!!!!!!

time elapsed:  1.7898292541503906

Process finished with exit code 0
```

וסים טנוס: 207866328
פארוק כריים: 314656869

branch and bound - LP relaxation:

עבודת הפונקציה: כמו שראינו בהרצאה, לוקחים את ה-DENSITE של כל פריט וממינים את המערך, וכל פעם ניקח את הכי גדולים מבחינת העדיפות, ואם לא נשאר מספיק מקום לפריט שלקחנו אותו לפי העדיפות אז ניקח את החלק המתאים ממינו ונצא.

```
def integralItemSizes(vector: list):
    estimate = ARGS.sacks[0].calculateValue(vector)
    capacities = [sack.capacity for sack in ARGS.sacks]
    currentWeights = [sack.calculateWeight(vector) for sack in ARGS.sacks]
    minimumCut = 1
    for object, density in ARGS.sumDensities.items():
        if object.name < len(vector):
            if vector[object.name] == 0:
                continue
        for i in range(len(currentWeights)):
            currentWeights[i] -= object.weight
            estimate += object.value

    for object, density in ARGS.sumDensities.items():
        if object.name < len(vector):
            continue
        flag = False
        for i in range(len(currentWeights)):
            if currentWeights[i] - capacities[i] < 0 or flag:
                flag = True
                ratio = abs(float((currentWeights[i] - capacities[i]) / object.weight))
                minimumCut = min(minimumCut, ratio)
            else:
                currentWeights[i] -= object.weight
        if flag:
            estimate += float(object.value * minimumCut)
            return estimate
        estimate += object.value
    return estimate
```

וסים טנוס: 207866328
פארוק כריים: 314656869

היוריסטיקת השק הלא חסום:

איך עובדת: כמו שראינו בכיתה, מתייחסים ל ESTIMATE כסכום כל הפרטים חוץ מאלו שלא לקחנו אותם (כלומר הם אפס בתוך הוקטור).

```
def unboundedWeight(vector: list):
    estimate = ARGS.sumValues
    for i in range(len(vector)):
        if vector[i] == 0:
            estimate -= ARGS.values[i]
    return estimate
```

- השתמשו בנירמול הפריטים לפי צפיפותם (Vi/Wi) ושימו לב לרגישות למיונם לפני תהליך החיפוש

את הצפיפות אנחנו מחשבים בתחילת הבעיה לכל השקים, ואחר כך ממינים בהתאם.

```
class MultiKnapsackProblem:
    def __init__(self, sacks: list, optimum: int, values: list):
        self.sacks = sacks
        self.optimum = optimum
        self.values = values
        self.bestValue = 0
        self.bestVector = []
        for sack in sacks:
            for i in range(len(values)):
                if sack.weights[i] == 0:
                    sack.density.append((float('inf'), values[i], sack.weights[i]))
                else:
                    sack.density.append((float(values[i] / sack.weights[i]), values[i], sack.weights[i]))
            sack.density.sort(reverse=True, key=lambda tup: tup[0])
```

וסים טנוס: 207866328
פארוק כריים: 314656869

- בדקו את שתי ההיוריסטיקות שהוצגו בשיעור, המזניחות את אילוצי הבעיה והשתמשו במוצלחת יותר:

1.1 שק לא חסום

1.2 משתני בחירה שבריים – מילוי השק בדיוק לפי גודלו תוך השלמתו בערך שברי.

לפי ההשוואות שלנו:
שתי היוריסטיקות מובילות לפתרון אופטימלי מכיוון שאנחנו בודקים את כל האפשרויות, אבל ההבדל ביניהם הוא רק מבחינת זמן ריצה.
כלומר יוריסטיקה שגוזמת ענפים כמה שיותר מוקדם, היא זו שתזכה בזמן ריצה מהיר יותר, וכמובן זה גם תלוי בסיבוכיות חישוב היוריסטיקה.
לפי התוצאות שלנו מצאנו שהיוריסטיקת השק הלא חסום היא יותר מהירה מכיוון שאנו מחשבים אותה בסיבוכיות לינארית באורך הקלט.
ולכן אנחנו נשתמש ביוריסטיקת השק הלא חסום.

דוגמת הרצה:
עבור השק ה-לא חסום:

```
vector:  [1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0] , value:  3800
vector:  [1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0] , value:  3900
wave:  6
vector:  [1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0] , value:  3960
vector:  [1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1] , value:  5275
wave:  7
vector:  [1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1] , value:  5435
vector:  [1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1] , value:  5550
wave:  8
vector:  [1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1] , value:  5590
wave:  9
vector:  [1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1] , value:  5630
vector:  [1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1] , value:  5730
vector:  [1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1] , value:  5890
vector:  [1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1] , value:  5930
vector:  [1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1] , value:  6090
wave:  10
vector:  [1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1] , value:  6110
wave:  11
vector:  [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1] , value:  6120

best solution found:
vector:  [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1] , value:  6120
optimum:  6120
reached optimum !!!!!!!!!

total time elapsed:    6.9637563228607  seconds.

Process finished with exit code 0
```

וסים טנוס: 207866328
פארוק כריים: 314656869

עבור שק חסום:

```
vector:  [1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0] , value:  3685
vector:  [1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0] , value:  3800
vector:  [1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0] , value:  3900
wave:  6
vector:  [1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0] , value:  3960
vector:  [1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1] , value:  5275
wave:  7
vector:  [1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1] , value:  5435
vector:  [1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1] , value:  5550
wave:  8
vector:  [1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1] , value:  5590
wave:  9
vector:  [1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1] , value:  5630
vector:  [1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1] , value:  5730
vector:  [1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1] , value:  5890
vector:  [1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1] , value:  5930
vector:  [1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1] , value:  6090
wave:  10
vector:  [1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1] , value:  6110
wave:  11
vector:  [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1] , value:  6120

best solution found:
vector:  [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1] , value:  6120
optimum:  6120
reached optimum !!!!!!!!!

total time elapsed:  8.290411710739136  seconds.

Process finished with exit code 0
```

אנחנו יכולים לראות מתוך הדוגמה הזאת שיש הבדיל בין שתי היוריסטיקות ושהיוריסטיקת השק הלא חסום היא יותר טובה.

וסים טנוס: 207866328
פארוק כריים: 314656869

חלק ב:

חלק ב': פתרון בעית CVRP

את האלגוריתם המיטבי שהתקבל בחלק א' שלבו בפתרון בעית הCVRP ממעבדה 2:

1. כאשר אתם מחלקים את פתרון הבעיה לשני שלבים מובחנים:

1.1 שלב ההקצאה – הקצאת לקוחות למשאיות באמצעות הפתרון המיטבי

1.2 שלב אופטימיזצית המסלולים

```
class CVRP:
    def __init__(self, cities, depot, distancMat, capacity):
        self.cities = cities
        self.depot = depot
        self.distancMat = distancMat
        self.vehicles = []
        self.capacity = capacity
        self.cost = 0

    def divideVehicles(self):
        while len(self.cities) > 0:
            vehicle = Vehicle(self.capacity, self.depot)
            self.vehicles.append(vehicle)
            vehicle.addCities(self.cities)
        for vehicle in self.vehicles:
            vehicle.printPath()

    def solveTsp(self, args):
        for vehicle in self.vehicles:
            tabuSearch(args, vehicle, self.distancMat, len(self.vehicles))

        for vehicle in self.vehicles:
            print(vehicle.bestPath, ', cost: ', vehicle.bestPathCost)
            self.cost += vehicle.bestPathCost
        print('Total Cost: ', self.cost)
```

לפי ה -CLASS הזה אנחנו מתארים איך עושים את זה, קודם מחלקים את המשאיות, ואז פותרים את בעיית TSP ע"י עזרה באלגוריתם OPT-2.

וסים טנוס: 207866328
פארוק כריים: 314656869

חלוקת המשאיות:

```python
def getNearestCity(self, cities):
    NeaerestCity, minDistance = None, float('inf')
    for city in cities:
        distanc = city.calculateDistance(self.centerX, self.centerY)
        if (distanc < minDistance):
            NeaerestCity = city
            minDistance = distanc
    return NeaerestCity

def addCity(self, cities):...


def addCities(self, cities):
    flag = True
    # while self.currValue > 0 and len(cities) > 0:  ## for the last vehicle
    while self.currValue > 0 and flag:  ## for the last vehicle
        # self.addCity(cities)
        flag = self.addCity(cities)
        self.updateCenter()

def getFarCity(self, cities):...

def updateCenter(self):
    city = self.cities[-1]
    x = city.x
    y = city.y
    self.centerX = float(((self.centerX * (len(self.cities) - 1)) + x) / len(self.cities))
    self.centerY = float(((self.centerY * (len(self.cities) - 1)) + y) / len(self.cities))
```

השם של כל פונקציה מתאר מה היא עושה, וכדי לתאר בדיוק מה עשינו ומאיפה הרעיון הזה נבע, מצורף לינק למאמר שפותר את הבעיה הזאת ושהמעבדה הנוכחית מתבסס על המאמר הזה:
([1811.07403.pdf (arxiv.org)](https://arxiv.org/pdf/1811.07403.pdf))
https://arxiv.org/pdf/1811.07403.pdf

תמונה מהמאמר - עמוד 3: (שהיא אותה תמונה של המעבדה)

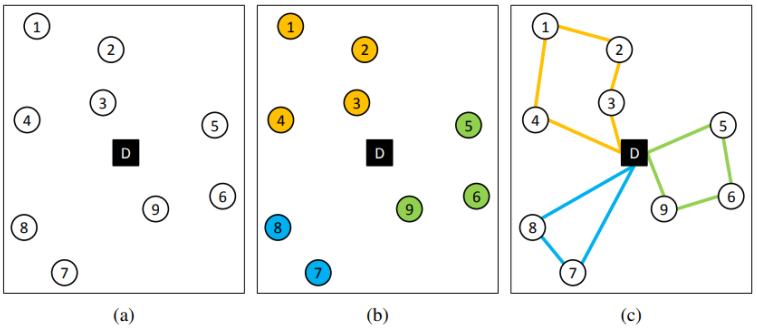


Figure 1: Overview of the CVRP and the 2-Phase-Heuristic. (a) Initial state with 9 customers and 1 depot. (b) Clustering phase results in three clusters found. (c) Routing phase determines shortest path inside each cluster.

וסים טנוס: 207866328
פארוק כריים: 314656869

תיאור שיטת החילוק:

לוקחים נקודה וקובעים אותה כמרכז של כל הערים שהמשאית רוצה לספק להם, ואם יש עוד נפח במשאית אנחנו נרחיב את קבוצת הערים (ע"י הוספת העיר הכי קרובה למרכז) ונשנה את המרכז בהתאם.

את בעיית ה-TSP:

פתרנו אותה לפי שיטת ה-OPT-2 שראינו ומימשנו במעבדה 2.

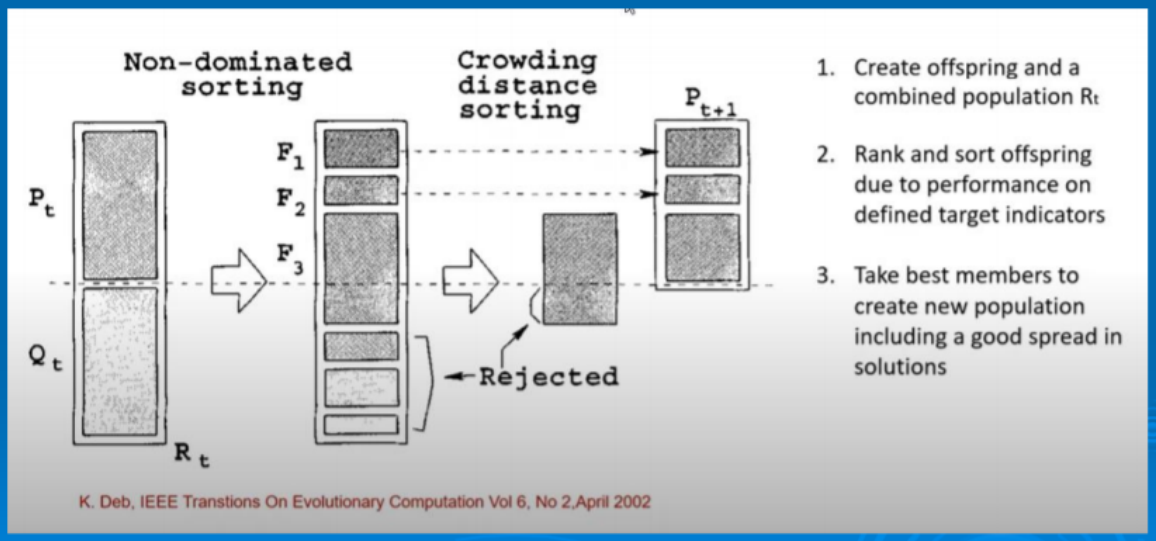
2. כאשר אתם מנסים לפתור את שתי הבעיות באופן סימולטני באמצעות האלגוריתם הגנטי **NSGA 2** ופונקצית מטרה multi objective function שתנסחו המשלבת בין פתרון שתיהן בו"ז.

עשינו את זה בדיוק כמו שראינו בהרצאה:
נעזרנו בסרטון קצר של 20 דקות שמסכם את האלגוריתם:
https://www.youtube.com/watch?v=SL-u_7hlqjA
The main class:

```
class GeneticAlgorithm:

    def __init__(self, args, CVRP):
        self.args = args
        self.crossoverType = args.CROSS
        self.mutationType = args.MUTATION
        self.selectionType = args.SELECTION
        self.population = []
        self.nextPopulation = []
        self.CVRP = CVRP
        self.w1 = 0.5
        self.w2 = 0.5
```

וסים טנוס: 207866328
פארוק כריים: 314656869

NSGA-II Main-Loop




```python
def run(self):
    startTime = time.time()
    self.initPopulation()
    paretoOptimal = []

    for _ in range(self.args.GA_MAXITER):
        iterTime = time.time()
        self.nextPopulation = []
        self.mate()
        fronts = self.nonDominatedSorting()
        paretoOptimal = fronts[0].members
        self.getNextPopulation(fronts)

        print('Generation time: ', time.time() - iterTime)
        print()
    print('Time elapsed: ', time.time() - startTime)
    return paretoOptimal
```

וסים טנוס: 207866328
פארוק כריים: 314656869

Non-dominated Sorting

- Assign each gene to its correct front
- Def: An individual **dominates** another individual iff it is better than the other in at least one indicator and not worse on all the other indicator(s)

```python
def nonDominatedSorting(self):
    size = len(self.nextPopulation)
    for i in range(size):
        for j in range(size):
            if i == j:
                continue
            self.domination(self.nextPopulation[i], self.nextPopulation[j])
    fronts = []
    while len(self.nextPopulation) > 0:
        front = Front()
        front.addMembers(self.nextPopulation)
        fronts.append(front)
    return fronts
```

וסים טנוס: 207866328
פארוק כריים: 314656869

Domination Example

- Assume the objective is to minimize indicator1 and indicator2 (two kpis):
- g1 = (x1,y1) , g2=(x2,y2)
- g1 dominates g2 iff:
- (x1<=x2 & y1<=y2) & (x1<x2 | y1 < y2)

```
def domination(self, member1: GAstruct, member2: GAstruct):
    if ((member1.pathCost <= member2.pathCost) and (member1.numberOfVehicles <= member2.numberOfVehicles)) and ((member1.pathCost < member
        member1.dominates.append(member2)
        member2.dominationCount += 1
```

```
def domination(self, member1: GAstruct, member2: GAstruct):
    if ((member1.pathCost <= member2.pathCost) and (member1.numberOfVehicles <= member2.numberOfVehicles)) |
            and ((member1.pathCost < member2.pathCost) or (member1.numberOfVehicles < member2.numberOfVehicles)):
        member1.dominates.append(member2)
        member2.dominationCount += 1
```

וסים טנוס: 207866328
פארוק כריים: 314656869

דוגמת הרצה:

```
Global Best:   --fitness = 202.72437563140326,  --number of vehicles = 4,  --path cost = 401.4487512628065,
Generation time:  0.15836763381958008

Global Best:   --fitness = 202.72437563140326,  --number of vehicles = 4,  --path cost = 401.4487512628065,
Generation time:  0.14991426467895508

Global Best:   --fitness = 202.72437563140326,  --number of vehicles = 4,  --path cost = 401.4487512628065,
Generation time:  0.15975260734558105

Global Best:   --fitness = 202.72437563140326,  --number of vehicles = 4,  --path cost = 401.4487512628065,
Generation time:  0.14379000663757324

Global Best:   --fitness = 202.72437563140326,  --number of vehicles = 4,  --path cost = 401.4487512628065,
Generation time:  0.15443658828735352

Global Best:   --fitness = 202.72437563140326,  --number of vehicles = 4,  --path cost = 401.4487512628065,
Generation time:  0.1547858715057373

Global Best:   --fitness = 202.72437563140326,  --number of vehicles = 4,  --path cost = 401.4487512628065,
Generation time:  0.15728163719177246

Global Best:   --fitness = 202.72437563140326,  --number of vehicles = 4,  --path cost = 401.4487512628065,
Generation time:  0.1545400619506836

Global Best:   --fitness = 202.72437563140326,  --number of vehicles = 4,  --path cost = 401.4487512628065,
Generation time:  0.15466952323913574

Global Best:   --fitness = 202.72437563140326,  --number of vehicles = 4,  --path cost = 401.4487512628065,
Generation time:  0.14968419075012207

Global Best:   --fitness = 202.72437563140326,  --number of vehicles = 4,  --path cost = 401.4487512628065,
Generation time:  0.14899468421936035
```

3. השוו את התוצאות של הפתרון הדו-שלבי מול הפתרון הסימולטני בטבלה בה כל שורה תתאים לבעיה ותבטא השוואה בין האלגוריתמים מבחינת מהירות ההתכנסות ואת איכות הפתרון שהתקבל. כמו כן תארו והסבירו את ההיוריסטיקות שהשתמשתם בהם לצורך הפתרון.

השוואות בין שני האלגוריתמים:

	1	2	3	4	5	6	7
OPTIMAL	375	835	521	832	735	817	1077
multi knapsack +TSP	385	964	618	952	827	907	1216
NSGA	395	887	635	905	790	930	1207

וסים טנוס: *207866328*
פארוק כריים: *314656869*

אנחנו יכולים לראות מתוך הטבלה ששני האלגוריתמים מבצעים עבודה טובה מאוד, וכמעט שניהם מחזירים אותן תוצאות.
מכיוון ששניהם עובדים באותו זמן אז מבחינת ההתכנסות שניהם עובדים באותה מהירות.
איכות: שתי השיטות מחזירות תוצאות קרובות, לפעמים זה סוטה יותר ולפעמים פחות, מה שאומר שמבחינת האיכות לשנים יש אותה איכות.

היוריסטיקה של השיטה הראשונה:
בוחרים את העיר הכי רחוקה, וקובעים אותה כמרכז של הערים שהמשאית הנוכחית הולכת לספק להם, ומסתכלים על העיר הכי קרובה למרכז הזה, אם יש עוד מקום במשאית לספק לה, אז מכניסים אותה לקבוצה ומשנים את המרכז בהתאים (לפי ההגדרה בתוך המאמר).
יתרון היוריסטיקה: בצורה הזאת אנו מחלקים את הערים בצורה הכי טובה כי כך אנחנו נדאג שהמשאית הנוכחית תטיל על ערים הכי קרובים אחד לשני ובכך אנחנו נשמור על זה שהמשאית לא תשלם על זה לספק לעיר רחוקה מדי.
היוריסטיקה של השיטה השנייה:
אנחנו נסתכל כל פעם על הגרף שלנו וננסה לתפוס פריתו אופטימה, לפי מספר המשאיות ומחיר המסלול שלנו, ומנסים למנמן את שניהם ביחד. כי מינימום משאיות זה אומר מינימום COST, ואנחנו מנסים למנמן את שניהם בו זמנית לפי ההרצאה.

לפי שני היתרונות מעלה, אנחנו נקבל פתרונות מאוד טובים ואיכותיים.