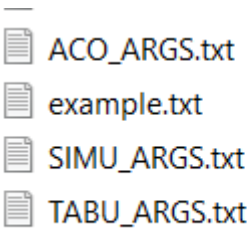
וסים טנוס: 207866328
פארוק כריים: 314656869

דוח מעבדה 2

הערה חשובה לבודק\ת:
נא לעשות extract לקובץ zip ולשים את תוכנית המain בתיקיה יחד עם תיקית INPUT וזה כדי שהmain יצליח לראות את התיקיה הזאת. הסיבה לכך היא שבקוד אנחנו משתמשים בpath שהוא יחסי למיקום המain.
כמו בתמונה:

בתוך תיקיית הinput יש קבצי הפרמטרים וניתן לשנות אותם לפני ההרצה.
שהם ACO-SIMU-TABU



ב genetic algorithm יש ui שמאפשר הזנת פרמטרים.

וסים טנוס: 207866328
פארוק כריים: 314656869

הערה לגבי האלגוריתמים:
בשלושת האלגוריתמים (tabu, simulated, aco) הם מזהים התכנסות ל local optimum . הם יודעים לאפס את הצמם, לשמור את הפתרון שהגיעו אליו ולהגריל מקום חדש להמשיך ממנו. הפתרון הטוב ביותר ייבחר כפתרון "אופטימלי".
את השיטה הזאת למדנו בקורס המבוא והיה כיף לנו להשתמש בה, (שזה סוג של BackTracking) וכמובן הצלחנו לשפר את הפתרון שקיבלנו.

וסים טנוס: 207866328
פארוק כריים: 314656869

חלק א:

1. לימדו את פורמט הקלט/פלט – (בעיה/פתרון) המוסברים בסעיפים בהמשך

```
class CVRP:

    def __init__(self, distanceMatrix, depot, cities, capacity, size):
        self.distanceMatrix = distanceMatrix
        self.cities = cities
        self.depot = depot
        self.capacity = capacity
        self.size = size
        self.best = []
        self.bestFitness = 0

    def calcPathCost(self, path):...

    def pathToVehicles(self, path):...

    def printSolution(self):
        print(self.bestFitness)
        paths = self.pathToVehicles(self.best)
        for path in paths:
            print(*path, sep=' ')
```

זהו הclass שדרכו אנו מייצגים את הבעיה, נסביר כל מה שנמצא בתוכו:

1-ה-CITIES: זה מערך של ערים, כך שעבור כל עיר יש את הנתונים שלה. (בנינו class מיוחד לזה)

וסים טנוס: 207866328
פארוק כריים: 314656869

```
class City:

    def __init__(self, id, x, y):
        self.id = id
        self.x = x
        self.y = y
        self.capacity = 0

    def setDemand(self, demand):
        self.capacity = demand
```

2- מטריצת מרחקים: כשקלטנו את הערים ואת הנקודות שלהן, חישבנו את המרחק בין כל עיר שמבחינתינו זו היא מטריצת הקשתות בתוך הגרף של הערים.

3-ה DEPOT: זה הוא המחסן הראשי והוא מסוג "עיר" כדי לייצג את מקומו וכל הפרטים הרלוונטים.

4- ה-CAPACITY: כל משאית כמה היא יכולה להעמיס/לספק.

5- ה-SIZE: מספר הערים שצריך לבקר בהם. (בלי דיבות)

6-ה-BEST ו-bestFitness: המסלול האופטימלי שמצאנו ואת המחיר שלו.

2. התאימו את בעית הדוגמא לעיל לפורמט הקלט ולכל אלגוריתם שאתם נדרשים לפתח

הראינו לעיל איך למדנו את הקלטים. עבור כל אלגוריתם שמימשנו, הוא מקבל מופע מ-CVRP שמכיל את כל המידע שאנו צריכים, וכל אלגוריתם פועל בהתאם. (נראה בהמשך הדוח את כל האלגוריתמים)

3. עבור האלגוריתמים השונים פתחו היוריסטיקות שונות שיכולות לסייע בפתרון – מותר כמובן לעשות שימוש בכל ההיוריסטיקות שנלמדו בקורס המבוא עבור בעיות TSP

היוריסטיקה ראשונה: היא להפוך את הבעיה לבעית KNAPSACK מוכללת, כך שקל משאית מבחינתינו היא שאק.
היוריסטיקה שפיתחנו היא:להסתכל על הבעיה פרמוטציה של הערים, ולהסתכל על כל השכיניים, וללכת לשכן שמקטין את אורך המסלול.
הסביר על השכיניים ואיך מיוצגים:
שכן- הוא פרמוטציה לכל הערים.

וסים טנוס: 207866328
פארוק כריים: 314656869

ייצוג- כל פרמוטציה מייצגת מסלולים שונים של מספר משאיות.

ייצוג מספר המשאיות בכל פרמוטציה:
נלך על הפרמוטציה מההתחלה, שזו המשאית הראשונה.
כל עוד היא יכולה לספק לעיר הבאה, היא תספק.
אם לא יכולה לספק עוד ויש עוד ערים, מתחילים במשאית חדשה.

4. קדדו את האלגוריתמים (ראו סעיף לגבי האלגוריתמים הספציפיים)

עבור כל אלגוריתם: נצרף תמונה לבסודו קוד ואת תרגום האלגוריתם:

TABU SEARCH:

```
sBest ← s0
bestCandidate ← s0
tabuList ← []
tabuList.push(s0)
while (not stoppingCondition())
    sNeighborhood ← getNeighbors(bestCandidate)
    bestCandidate ← sNeighborhood[0]
    for (sCandidate in sNeighborhood)
        if ( (not tabuList.contains(sCandidate)) and (fitness(sCandidate) > fitness(bestCandidate)) )
            bestCandidate ← sCandidate
        end
    end
    if (fitness(bestCandidate) > fitness(sBest))
        sBest ← bestCandidate
    end
    tabuList.push(bestCandidate)
    if (tabuList.size > maxTabuSize)
        tabuList.removeFirst()
    end
end
return sBest
```

וסים טנוס: 207866328
פארוק כריים: 314656869

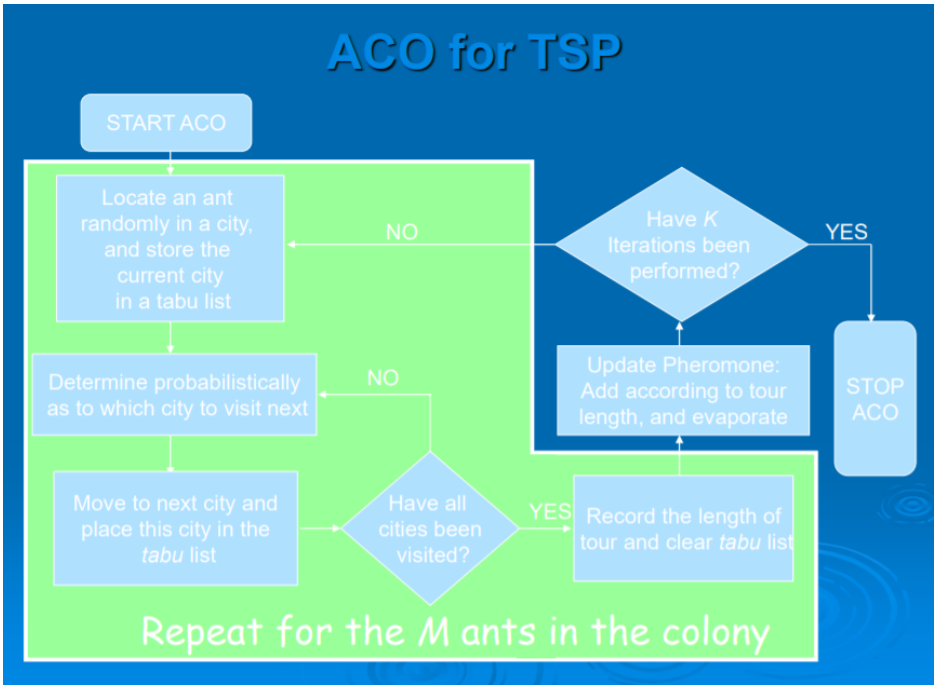
```python
def tabuSearch(problem, args):
    startTime = time.time()
    best = initGreedySol(problem.size, problem)
    bestFitness, _ = problem.calcPathCost(best)
    bestCandidate = best
    globalBest = best
    globalFitness = bestFitness
    tabuDict = {str(best): True}
    tabu = [best]
    local_counter = 0
    for _ in range(args.maxIter):
        iterTime = time.time()
        neighborhood = getNeighborhood(bestCandidate, args.numNeighbors)
        minimum, _ = problem.calcPathCost(neighborhood[0])
        bestCandidate = neighborhood[0]
        for neighbor in neighborhood:
            cost, _ = problem.calcPathCost(neighbor)
            if cost < minimum and not tabuDict.get(str(neighbor), False):
                minimum = cost
                bestCandidate = neighbor
```

וסים טנוס: 207866328
פארוק כריים: 314656869

```
        if minimum < bestFitness:
            bestFitness = minimum
            best = bestCandidate
            local_counter = 0
        elif minimum == bestFitness:
            local_counter += 1
        if bestFitness < globalFitness:
            globalBest = best
            globalFitness = bestFitness
        tabu.append(bestCandidate)
        tabuDict[str(bestCandidate)] = True
        if len(tabu) > args.maxTabu:
            tabuDict[str(tabu[0])] = False
            tabu.pop(0)
        if local_counter == args.localOptStop:
            if bestFitness < globalFitness:
                globalBest = best
                globalFitness = bestFitness
            bestCandidate = initGreedySol(problem.size, problem)
            best = bestCandidate
            bestFitness, _ = problem.calcPathCost(best)
            local_counter = 0
            tabuDict = {str(bestCandidate): True}
        print('Generation time: ', time.time() - iterTime)
        print('sol = ', best)
        print('cost = ', bestFitness)
        print()
    print('Time elapsed: ', time.time() - startTime)
    problem.best = globalBest
    problem.bestFitness = globalFitness
```

וסים טנוס: 207866328
פארוק כריים: 314656869

ACO:



וסים טנוס: 207866328
פארוק כריים: 314656869

```python
def ACO(problem, args):
    startTime = time.time()
    pheremonMatrix = [[float(1000) for _ in range(problem.size)] for _ in range(problem.size)]
    bestPath = []
    bestFitness = float('inf')
    currentBestPath = []
    currentBestFitness = float('inf')
    globalBest = []
    globalFitness = float('inf')
    local_counter = 0
    for _ in range(args.maxIter):
        iterTime = time.time()
        tempPath = getPath(problem, pheremonMatrix, args)
        tempFitness, _ = problem.calcPathCost(tempPath)
        if tempFitness < currentBestFitness:
            currentBestFitness = tempFitness
            currentBestPath = tempPath
        if currentBestFitness < bestFitness:
            bestFitness = currentBestFitness
            bestPath = currentBestPath
            local_counter = 0
        if currentBestFitness == bestFitness:
            local_counter += 1
        updatePheremons(pheremonMatrix, tempPath, tempFitness, args.q, args.p)
        print('Generation time: ', time.time() - iterTime)
        print('sol = ', bestPath)
        print('cost = ', bestFitness)
        print()
```

```python
    if local_counter == args.localOptStop:
        pheremonMatrix = [[float(1000) for _ in range(problem.size)] for _ in range(problem.size)]
        local_counter = 0
        if bestFitness < globalFitness:
            globalBest = bestPath
            globalFitness = bestFitness
        bestPath = []
        bestFitness = float('inf')
        currentBestPath = []
        currentBestFitness = float('inf')
print('Time elapsed: ', time.time() - startTime)
problem.best = globalBest
problem.bestFitness = globalFitness
```

וסים טנוס: 207866328
פארוק כריים: 314656869

SIMULATED ANNEALING:

```
initialize (temperature T, random starting point)
while cool_iteration <= max_iterations
    cool_iteration = cool_iteration + 1
    temp_iteration = 0
    while temp_iteration <= nrep
        temp_iteration = temp_iteration + 1
        select a new point from the neighborhood
        compute current_cost (of this new point)
        δ = current_cost - previous_cost
        if δ<0, accept neighbor
        else, accept with probability exp(-δ/T)
    end while
    T = α * T        (0<α<1)
end while
```

וסים טנוס: 207866328
פארוק כריים: 314656869

```python
def simulatedAnnealing(problem, args):
    startTime = time.time()
    best = initGreedySol(problem.size, problem)
    bestFitness, _ = problem.calcPathCost(best)
    globalBest = best
    globalFitness = bestFitness
    currentBest = best
    currentFitness = bestFitness
    temperature = float(args.temperature)
    local_counter = 0
    for _ in range(args.maxIter):
        iterTime = time.time()
        LK = 30
        neighborhood = getNeighborhood(best, args.numNeighbors)
        for _ in range(LK):
            randNeighbor = neighborhood[randint(0, len(neighborhood) - 1)]
            neighborFitness, _ = problem.calcPathCost(randNeighbor)
            diff = neighborFitness - bestFitness
            metropolis = float(exp(float(-1 * diff) / temperature))
            if neighborFitness < currentFitness or rand() < metropolis:
                currentFitness = neighborFitness
                currentBest = randNeighbor
        if currentFitness < bestFitness:
            best = currentBest
            bestFitness = currentFitness
            local_counter = 0
        if currentFitness == bestFitness:
            local_counter += 1
        if bestFitness < globalFitness:
            globalBest = best
            globalFitness = bestFitness
```

וסים טנוס: 207866328
פארוק כריים: 314656869

```
        if local_counter == args.localOptStop:
            if bestFitness < globalFitness:
                globalBest = best
                globalFitness = bestFitness
            best = initGreedySol(problem.size, problem)
            bestFitness, _ = problem.calcPathCost(best)
            currentBest = best
            currentFitness = bestFitness
            local_counter = 0
            temperature = float(args.temperature)
        print('Generation time: ', time.time() - iterTime)
        print('sol = ', best)
        print('cost = ', bestFitness)
        print()
        temperature *= args.alpha
    print('Time elapsed: ', time.time() - startTime)
    problem.best = globalBest
    problem.bestFitness = globalFitness
```

וסים טנוס: 207866328
פארוק כריים: 314656869

GA:

פסאודו קוד כמו מעבדה 1.

```
def run(self):
    startTime = time.time()
    self.initPopulation()

    repeat = 0
    bestFitness = float('inf')

    found = False

    best = []
    genBestFit = 0

    for _ in range(self.args.GA_MAXITER):
        iterTime = time.time()
        self.calcFitness()
        self.sortByFitness()
        self.printBest()
        self.calcAvgSd()

        # this checks if we have reached a local optimum or found the goal
        if repeat == self.args.LOCAL_STOP_ITER or self.population[0].getFitness() == 0:...

        best = self.population[0].getString()[:]
        genBestFit = self.population[0].getFitness()

        if bestFitness == genBestFit:
            repeat += 1
        elif genBestFit < bestFitness:
            bestFitness = genBestFit
            repeat = 1

        try:
            self.mate()
        except:
            print('Generation time: ', time.time() - iterTime)
            break
        self.swap()
        self.aging()
        print('Generation time: ', time.time() - iterTime)
        print()
    print('Time elapsed: ', time.time() - startTime)
    self.CVRP.best = best
```

וסים טנוס: 207866328
פארוק כריים: 314656869

5. עבור כל היוריסטיקה בה אתם עושים שימוש הסבירו את יתרונותיה ביחס לבעית הCVRP

יתרון ההיוריסטיקה: מכיוון שאנחנו מחפשים בהיוריסטיקה שלנו מסלול משפר את עלות המסלולים של המשאיות, זה יוביל ישאף לפתרון האופטימלי שהוא בעצם מסלול קצר ביותר שדרכו אנחנו יכולים לספק לכל הערים את מה שהם רוצים.

6. שלבו את ההיוריסטיקות באלגוריתמים המתאימים, בצעו "בדיקת שפיות" על הדוגמא לעיל תוך בחינת ההיוריסטיקות השונות– ותוך כמה זמן. הדפיסו את הפתרון עצמו לפי מטריצת הפלט ובנוסף את זמני הריצה (ELAPSED וCPU).

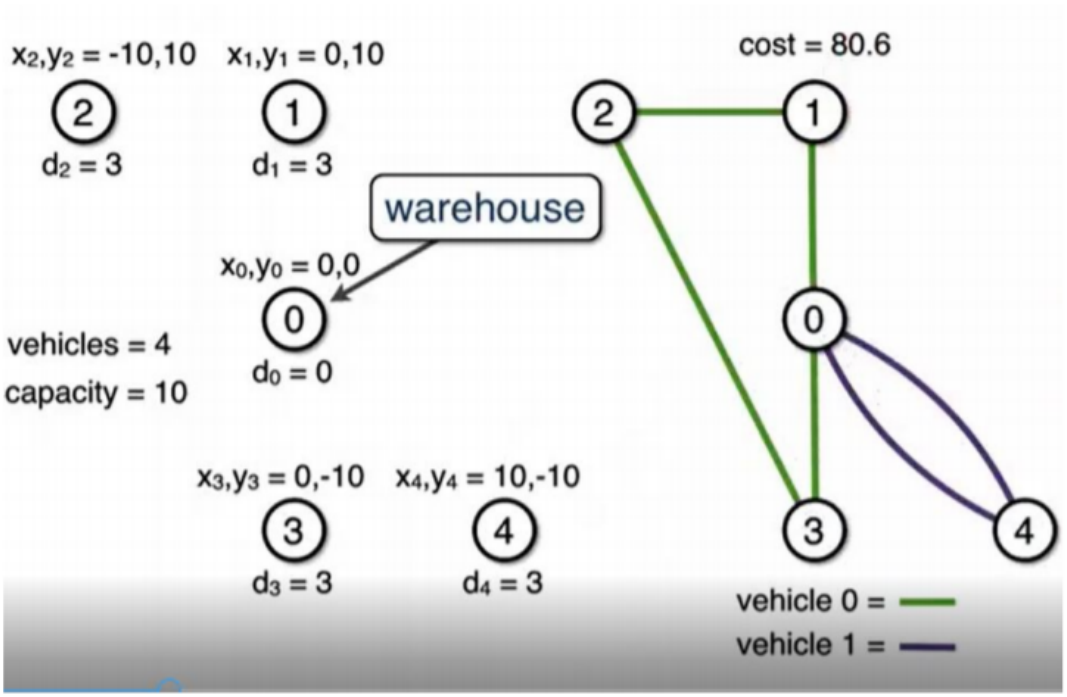
לפי מה שאמר לנו שי, אנו צריכים לעשות בדיקת שפיות על הבעיה בקובץ המטלה.

$\overline{i \in V}$

לדוגמא: בהנתן קואורדינטות המחסן (0,0), קואורדינטות הערים מסומנות בx,y הביקוש בכל עיר בd מציאת מסלול, (תת-אופטימלי), ל4 רכבים כשתכולת כל רכב היא 10 בעזרת 2 רכבים בלבד:

(a) הרכב הראשון יוצא מהמחסן מבקר בערים 1,2,3 וחוזר למחסן

(b) הרכב השני יוצא מהמחסן מבקר בעיר 4 וחוזר



וסים טנוס: 207866328
פארוק כריים: 314656869

TABU:

```
Generation time:  0.016954421997070312
sol =  [1, 4, 3, 2]
cost =  80.6449510224598

Generation time:  0.012966394424438477
sol =  [1, 4, 3, 2]
cost =  80.6449510224598

Generation time:  0.013927698135375977
sol =  [1, 4, 3, 2]
cost =  80.6449510224598

Generation time:  0.014601469039916992
sol =  [1, 2, 3, 4]
cost =  80.6449510224598

Generation time:  0.02393507957458496
sol =  [1, 2, 3, 4]
cost =  80.6449510224598

Generation time:  0.01795220375061035
sol =  [2, 1, 4, 3]
cost =  80.6449510224598

Generation time:  0.018949031829833984
sol =  [2, 1, 4, 3]
cost =  80.6449510224598

Generation time:  0.020125389099121094
sol =  [2, 1, 4, 3]
cost =  80.6449510224598

Time elapsed:  0.323927640914917
Tabu Search
80.6449510224598
0 2 1 4 0
0 3 0

Process finished with exit code 0
```

וסים טנוס: 207866328
פארוק כריים: 314656869

SIMULATED ANNEALING:

```
Generation time:  0.0059850215911865234
sol =  [2, 1, 3, 4]
cost =  82.42640687119285

Generation time:  0.005983114242553711
sol =  [1, 2, 3, 4]
cost =  80.6449510224598

Generation time:  0.004987239837646484
sol =  [4, 3, 2, 1]
cost =  80.6449510224598

Generation time:  0.005982160568237305
sol =  [2, 3, 4, 1]
cost =  80.6449510224598

Generation time:  0.00498652458190918
sol =  [2, 1, 4, 3]
cost =  80.6449510224598

Generation time:  0.0059854984283447266
sol =  [2, 1, 4, 3]
cost =  80.6449510224598

Generation time:  0.005984306335449219
sol =  [4, 1, 2, 3]
cost =  80.6449510224598

Generation time:  0.006981611251831055
sol =  [3, 2, 1, 4]
cost =  80.6449510224598

Time elapsed:  0.1156916618347168
Simulated Annealing
80.6449510224598
0 3 2 1 0
0 4 0

Process finished with exit code 0
```

וסים טנוס: 207866328
פארוק כריים: 314656869

ACO:

```
Generation time:  0.0
sol =  [3, 4, 1, 2]
cost =  80.6449510224598

Generation time:  0.000997781753540039
sol =  [3, 4, 1, 2]
cost =  80.6449510224598

Generation time:  0.0
sol =  [3, 4, 1, 2]
cost =  80.6449510224598

Generation time:  0.0
sol =  [3, 4, 1, 2]
cost =  80.6449510224598

Generation time:  0.0
sol =  [3, 4, 1, 2]
cost =  80.6449510224598

Generation time:  0.0
sol =  [3, 4, 1, 2]
cost =  80.6449510224598

Generation time:  0.0
sol =  [3, 4, 1, 2]
cost =  80.6449510224598

Generation time:  0.0
sol =  [3, 4, 1, 2]
cost =  80.6449510224598

Time elapsed:  0.004015684127807617
ACO
80.6449510224598
0 3 4 1 0
0 2 0

Process finished with exit code 0
```

וסים טנוס: 207866328
פארוק כריים: 314656869

GA:

```
Best:  [2, 3, 4, 1]  ( 80.6449510224598 ) fitness data:   avarage:  84.22588820828906  || standard deviation: 5.728815655208648
Generation time:  0.30569028854370117

Best:  [2, 3, 4, 1]  ( 80.6449510224598 ) fitness data:   avarage:  84.13883529167285  || standard deviation: 5.656837574297271
Generation time:  0.3467090129852295

Best:  [2, 3, 4, 1]  ( 80.6449510224598 ) fitness data:   avarage:  84.30951519119982  || standard deviation: 5.792870865389278
Generation time:  0.26005053520202637

Best:  [2, 3, 4, 1]  ( 80.6449510224598 ) fitness data:   avarage:  84.10153862215958  || standard deviation: 5.658788654795661
Generation time:  0.2483196258544922

Best:  [2, 3, 4, 1]  ( 80.6449510224598 ) fitness data:   avarage:  84.3218000565389  || standard deviation: 5.7722040981880625
Generation time:  0.23885893821716309

Best:  [2, 3, 4, 1]  ( 80.6449510224598 ) fitness data:   avarage:  84.08359257439267  || standard deviation: 5.621524073234588
Generation time:  0.25830984115600586

Best:  [2, 3, 4, 1]  ( 80.6449510224598 ) fitness data:   avarage:  84.20604204079764  || standard deviation: 5.715737183544595
Generation time:  0.2663004398345947

Best:  [2, 3, 4, 1]  ( 80.6449510224598 ) fitness data:   avarage:  84.23687544414689  || standard deviation: 5.768207721496991
Generation time:  0.27426624298095703

Best:  [2, 3, 4, 1]  ( 80.6449510224598 ) fitness data:   avarage:  84.2440481450523  || standard deviation: 5.73515478016716
Generation time:  0.26294422149658203

Best:  [2, 3, 4, 1]  ( 80.6449510224598 ) fitness data:   avarage:  84.4338647773293  || standard deviation: 5.856882193985541
Generation time:  0.012966632843017578

found.
Time elapsed:  6.149881601333618
Genetic Algorithm
80.6449510224598
0 2 3 4 0
0 1 0

Process finished with exit code 0
```

וסים טנוס: 207866328
פארוק כריים: 314656869

חלק ב:

נתחיל בכך שמה שנמצא בסעיף "7" משמש אותנו לכל הסעיפים 7-12.

תוכן: השקענו המון זמן (מעל 10 שעות) בהרצת כל בעיה על כל אלגוריתם עשרה (10) פעמים.

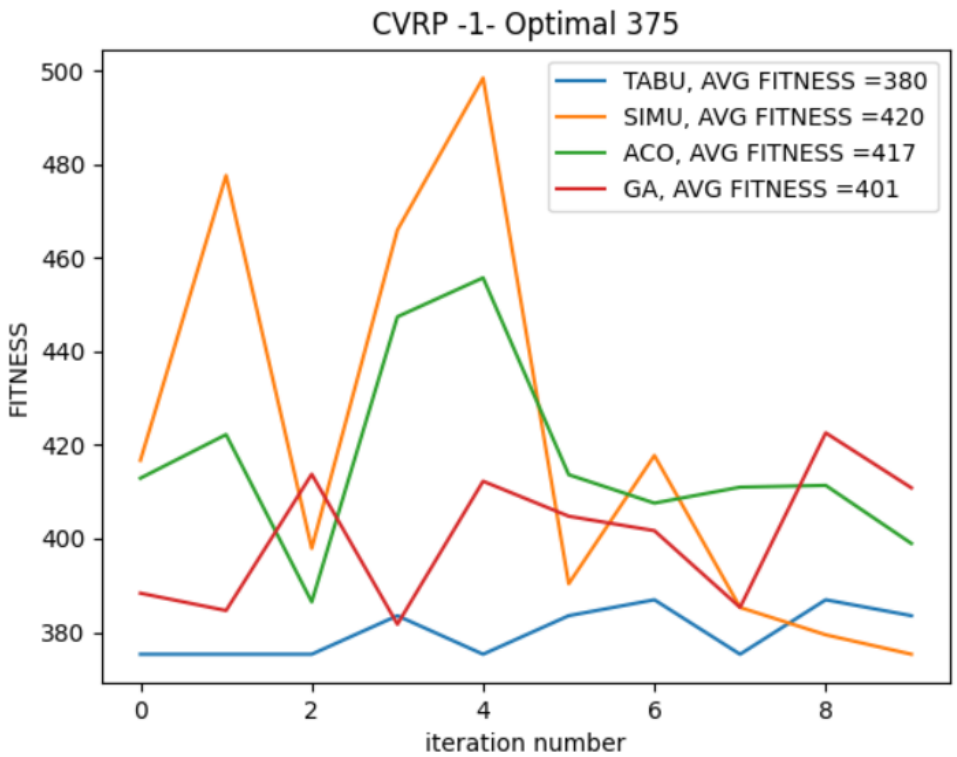
****יופיע 14 גרפים- 7 גרפים שיצרנו ע"י הרצת אלגוריתמים עם קונפיגורציה התחלתית רנדומלית. ועוד 7 גרפים שבנינו ע"י הרצה עם קונפיגורציה התחלתית טובה, ובכך נצליח להראות שיפור משמעותי.**

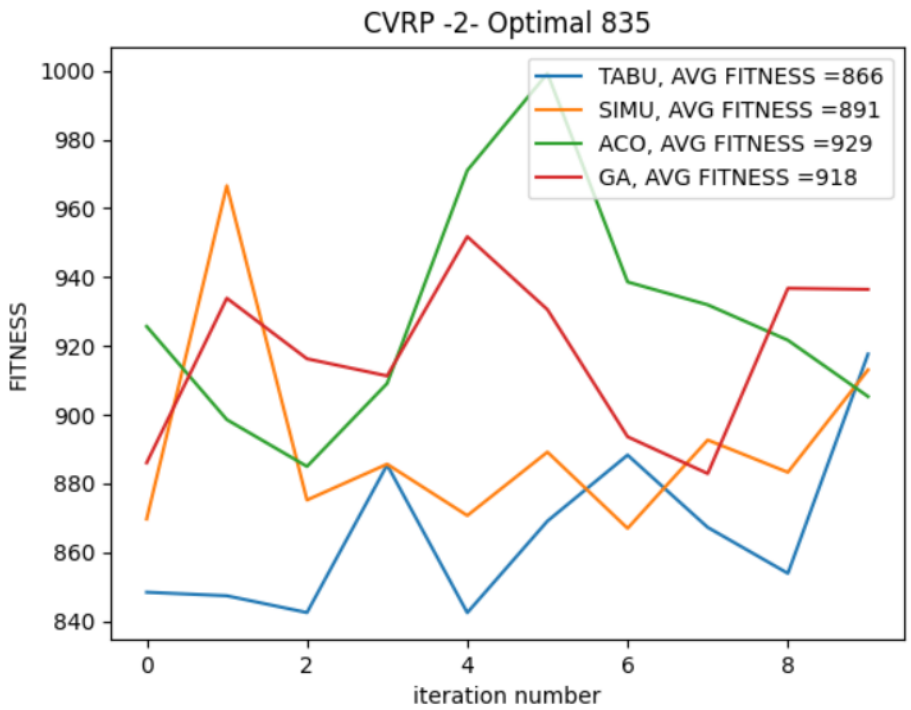
7. הריצו את האלגוריתמים על הבעיות המופיעות בסעיף הקלט. הקציבו זמן קבוע כרצונכם כבסיס השוואתי בין האלגוריתמים והדפיסו את הפתרונות.

בכל גרף מצוין שם האלגוריתם וכמה ה-FITNESS שלו בכל איטרציה ואת הממוצע של כל האיטרציות.
עבור הזמן: במקום הגבלת זמן, הגבלנו את מספר הצעדים בכל ריצה (שהוא 1500 צעדים).

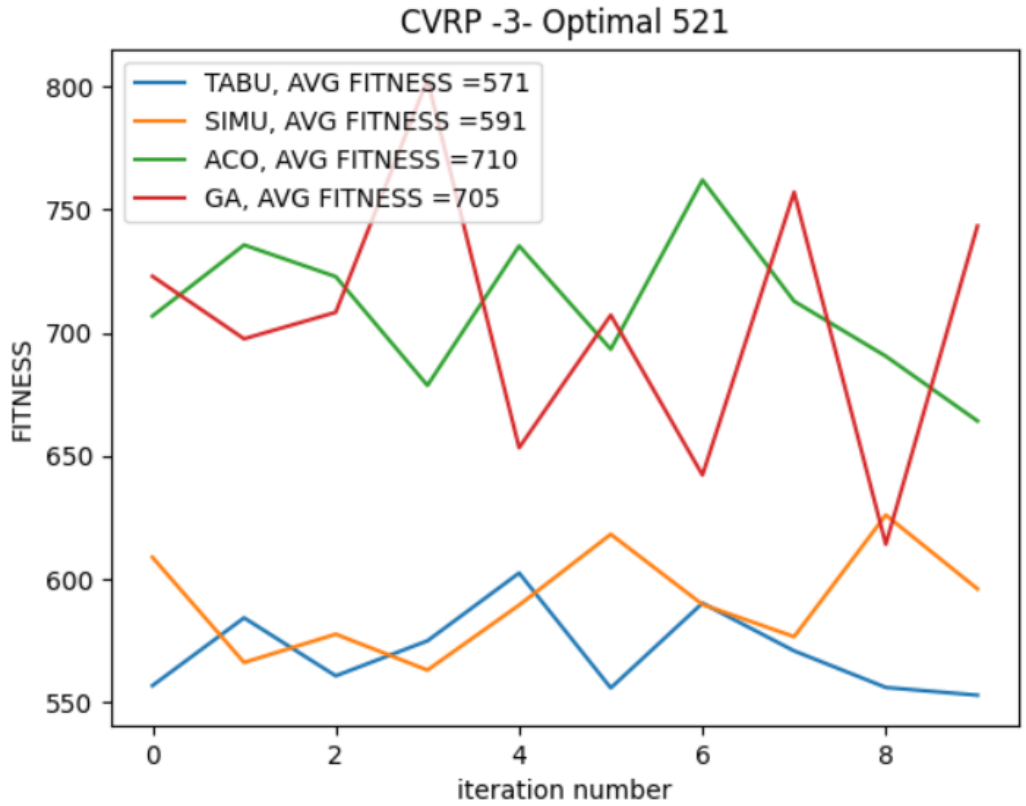
וסים טנוס: 207866328
פארוק כריים: 314656869

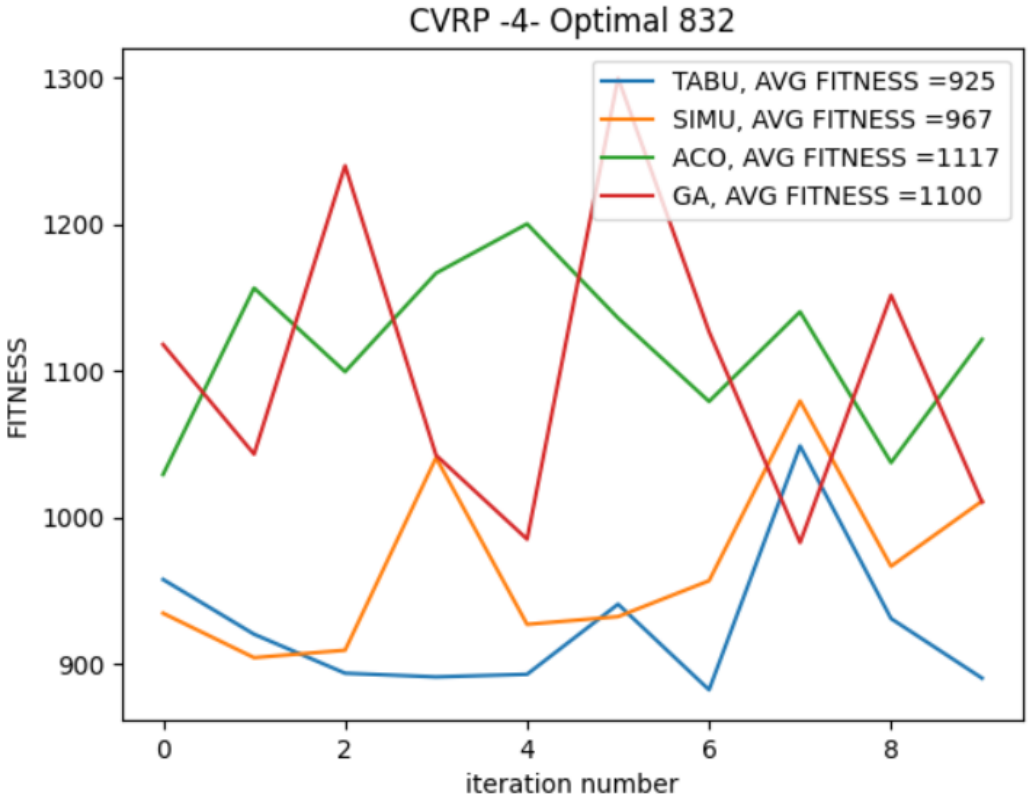
הגרפים עם קונפיגורציה התחלתית רנדומלית:



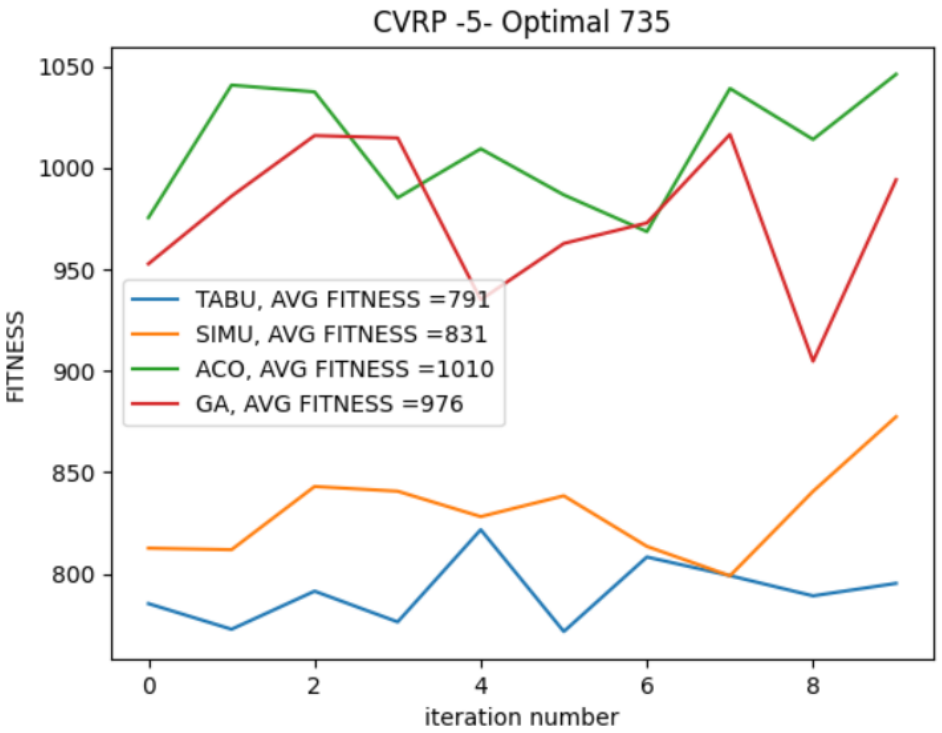


וסים טנוס: 207866328
פארוק כריים: 314656869

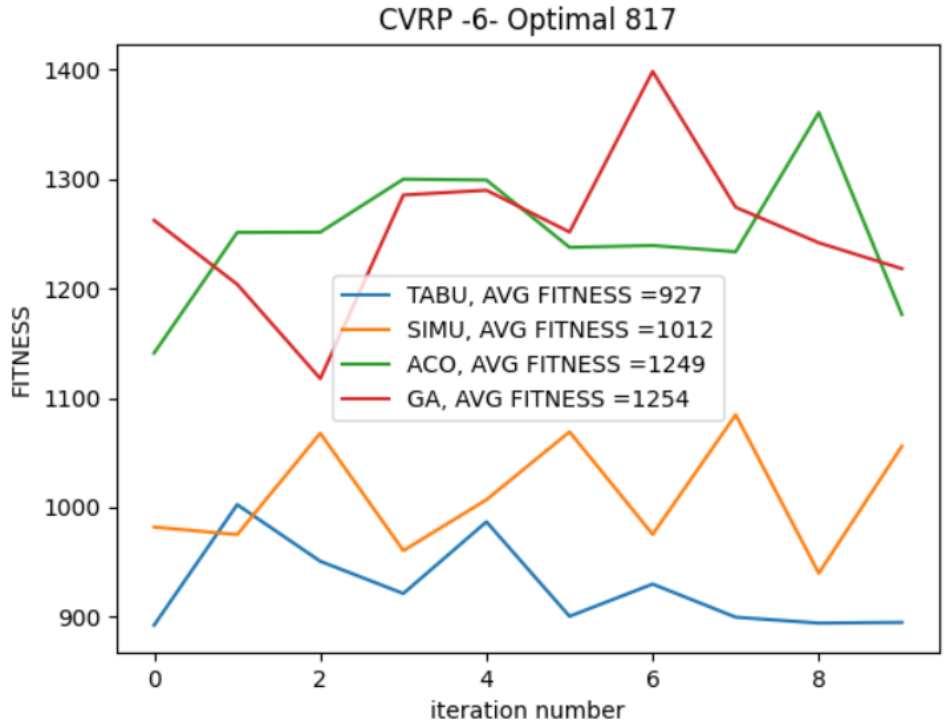


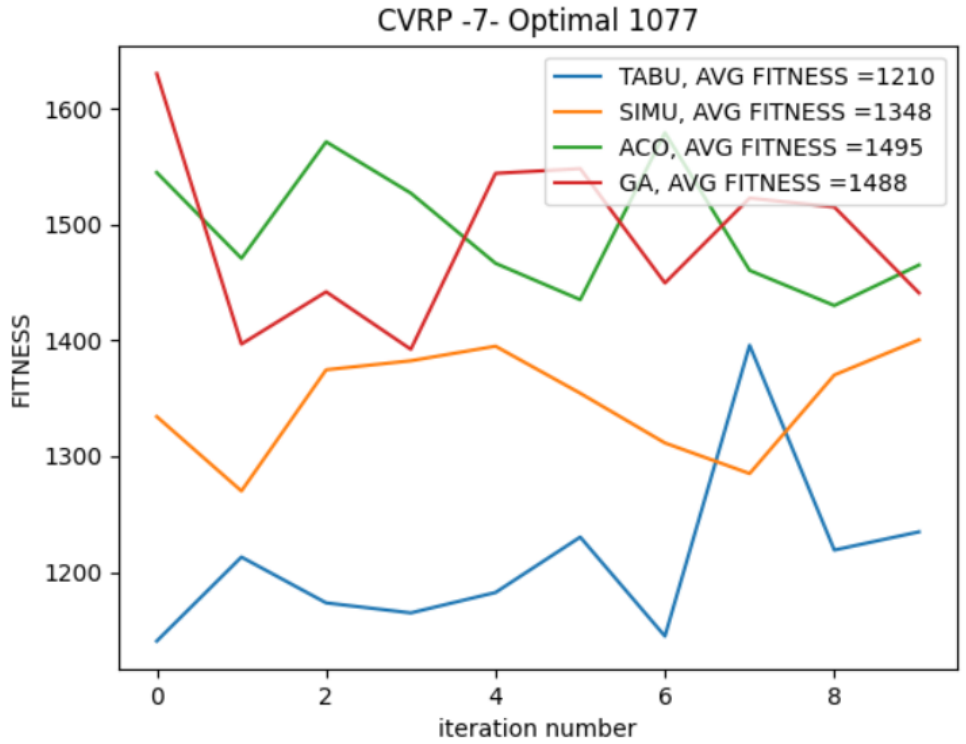


וסים טנוס: 207866328
פארוק כריים: 314656869



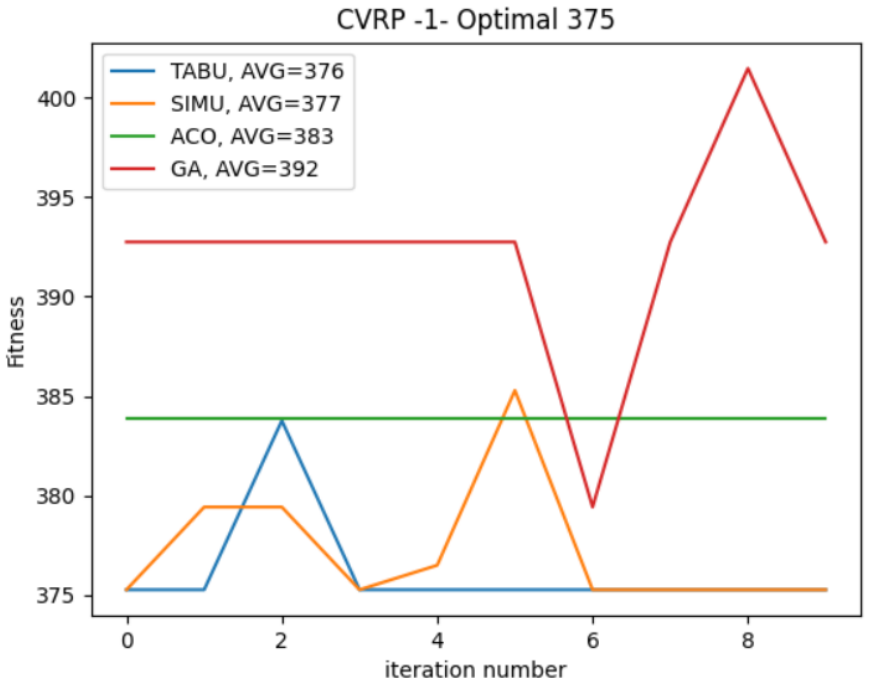
וסים טנוס: 207866328
פארוק כריים: 314656869

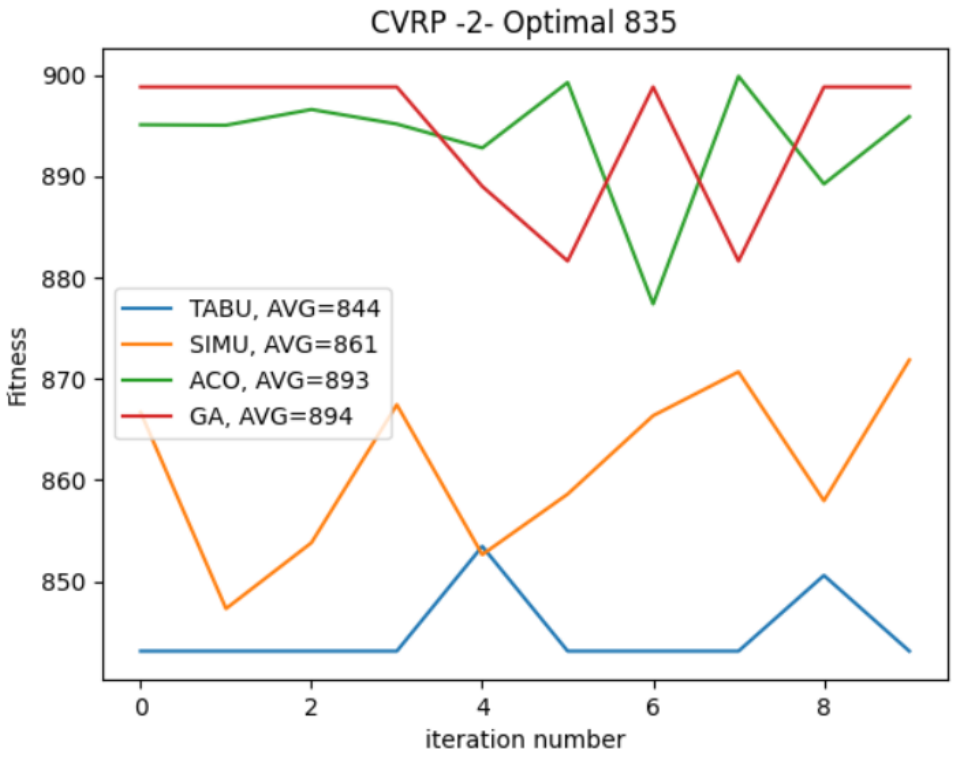




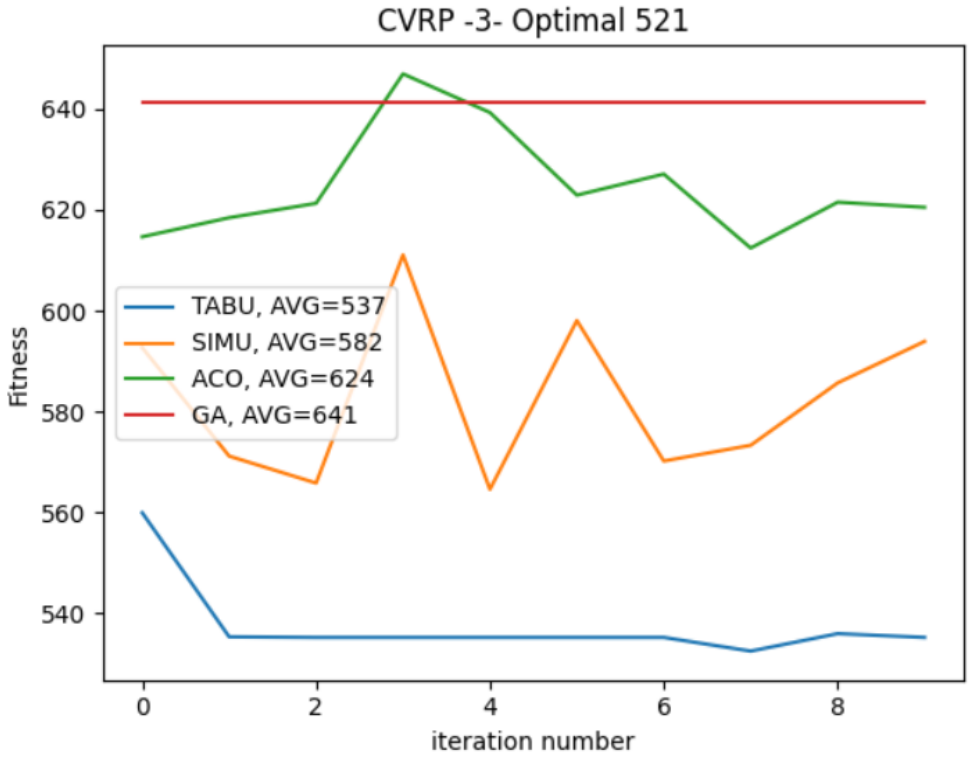
וסים טנוס: 207866328
פארוק כריים: 314656869

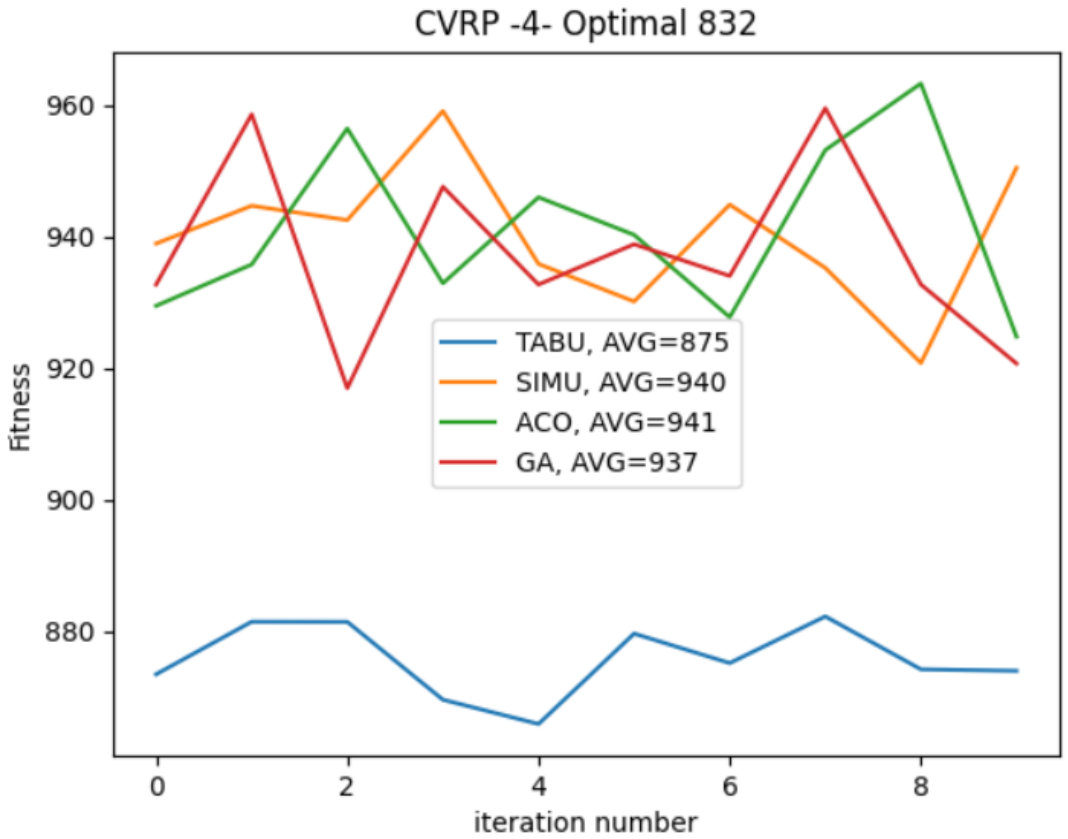
גרפים עם קונפיגורציה התחלתית טובה:



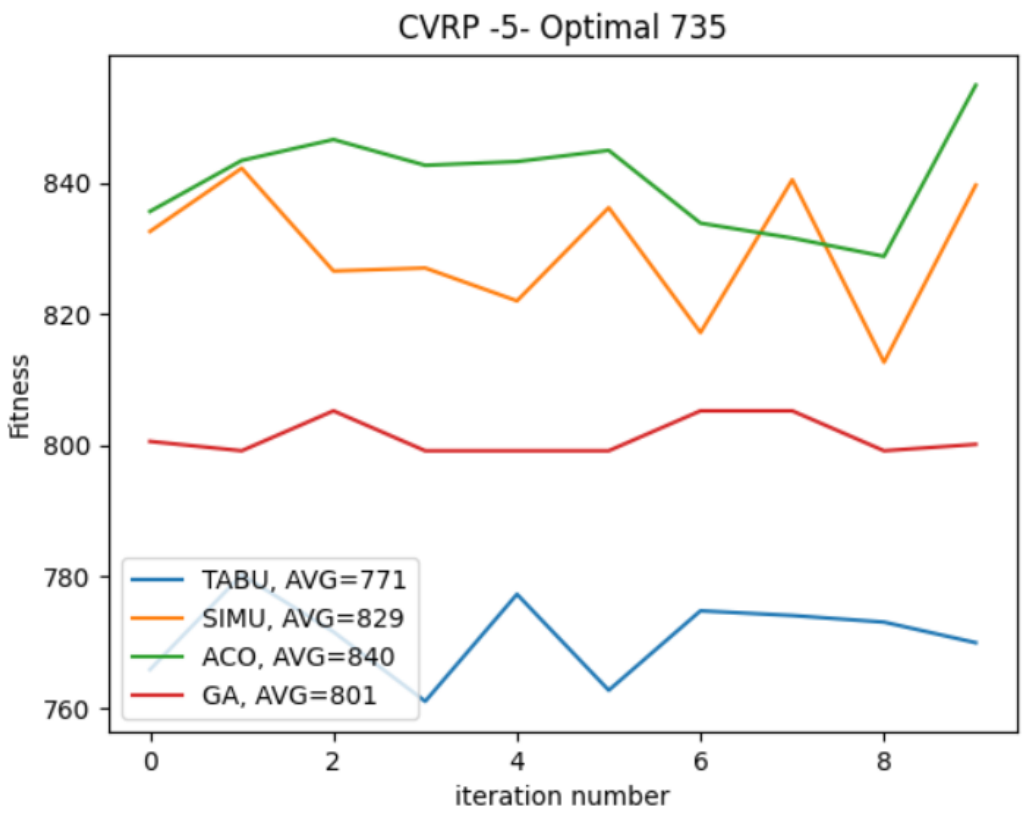


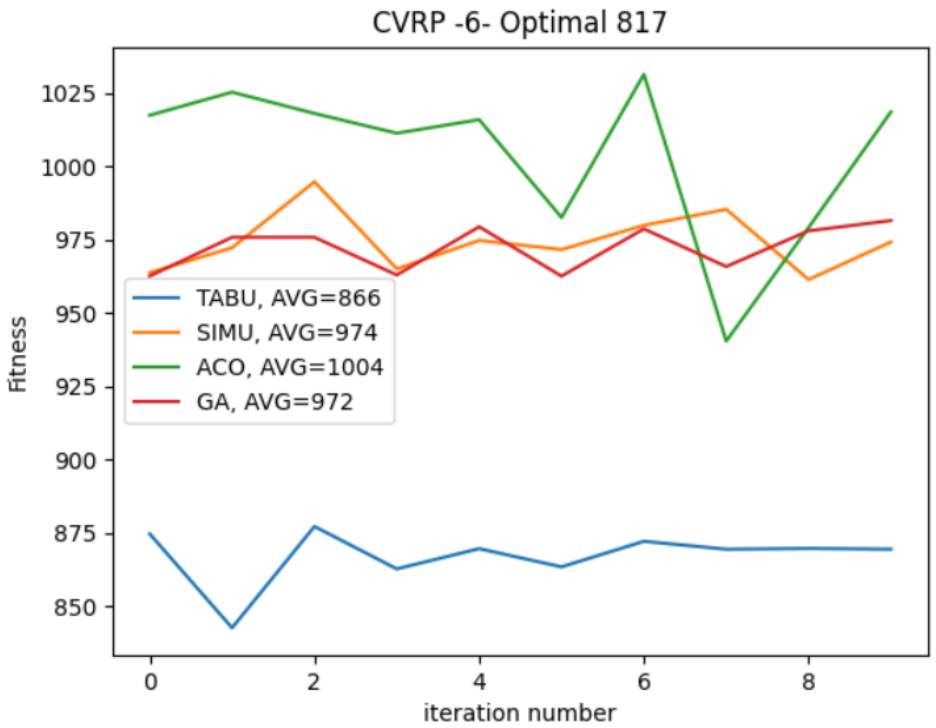
וסים טנוס: 207866328
פארוק כריים: 314656869



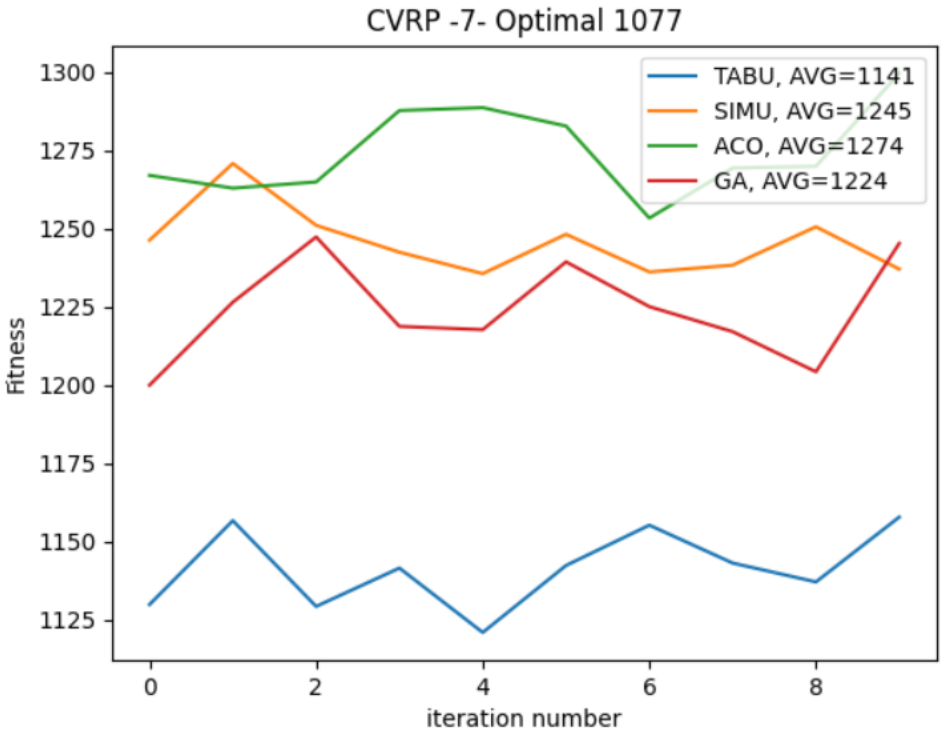


וסים טנוס: 207866328
פארוק כריים: 314656869





וסים טנוס: 207866328
פארוק כריים: 314656869



נמשיך עם 7 הגרפים הטובים:

8. את התוצאות רכזו בטבלה השוואתית. שורה נפרדת לזמספר הבעיה,הפתרון האופטימלי הידוע עבורה, שם האלגוריתם, האם מצא פתרון, מה ערכו, מה מרחקו מהאופטימלי, בכמה זמן CPU וELAPSED רץ.

עבור כל גרף של כל בעיה שבסעיף "7" , בנינו טבלה מסכמת.

בעיה 1: אופטימלי 375

	Tabu	Simul	ACO	GA
מצא פתרון	כן	כן	כן	כן
ערכו [בממוצע]	376	377	383	392
מרחק מ- OPT	1	2	8	17
זמן (sec) לכל איטרציה	0.045	0.007	0.002	0.3
זמן כולל =זמן	67.5	10.5	3	450

וסים טנוס: 207866328
פארוק כריים: 314656869

לכל איטרציה * 1500				

בעיה 2: אופטימלי 835

	Tabu	Simul	ACO	GA
מצא פתרון	כן	כן	כן	כן
ערכו [בממוצע]	844	861	893	894
מרחק מ- OPT	9	26	58	59
זמן (sec) לכל איטרציה	0.06	0.01	0.004	0.44
זמן כולל =זמן לכל איטרציה * 1500	90	15	6	660

בעיה 3: אופטימלי 521

	Tabu	Simul	ACO	GA
מצא פתרון	כן	כן	כן	כן
ערכו [בממוצע]	537	582	624	641
מרחק מ- OPT	16	61	103	120
זמן (sec) לכל איטרציה	0.09	0.01	0.01	0.5
זמן כולל =זמן לכל איטרציה * 1500	135	15	15	750

וסים טנוס: 207866328
פארוק כריים: 314656869

בעיה 4: אופטימלי 832

	Tabu	Simul	ACO	GA
מצא פתרון	כן	כן	כן	כן
ערכו [בממוצע]	875	940	941	937
מרחק מ- OPT	43	108	109	105
זמן (sec) לכל איטרציה	0.02	0.01	0.024	0.6
זמן כולל =זמן לכל איטרציה * 1500	30	15	36	900

בעיה 5: אופטימלי 735

	Tabu	Simul	ACO	GA
מצא פתרון	כן	כן	כן	כן
ערכו [בממוצע]	771	829	840	801
מרחק מ- OPT	36	94	105	66
זמן (sec) לכל איטרציה	0.15	0.12	0.025	0.65
זמן כולל =זמן לכל איטרציה * 1500	225	180	37.5	975

וסים טנוס: 207866328
פארוק כריים: 314656869

בעיה 6: אופטימלי 817

	Tabu	Simul	ACO	GA
מצא פתרון	כן	כן	כן	כן
ערכו [בממוצע]	866	974	1004	972
מרחק מ- OPT	49	157	187	155
זמן (sec) לכל איטרציה	0.16	0.02	0.04	0.75
זמן כולל =זמן לכל איטרציה * 1500	240	30	60	1125

בעיה 7: אופטימלי 1077

	Tabu	Simul	ACO	GA
מצא פתרון	כן	כן	כן	כן
ערכו [בממוצע]	1141	1245	1274	1224
מרחק מ- OPT	64	168	197	147
זמן (sec) לכל איטרציה	0.2	0.02	0.04	0.8
זמן כולל =זמן לכל איטרציה * 1500	300	30	60	1200

וסים טנוס: 207866328
פארוק כריים: 314656869

9. ציינו בטבלה נפרדת את סיבוכיות המקום של האלגוריתמים

חשוב לציין ש- (neighbors + tabu-list+population) בשליטת המשתמש.

GA	ACO	Simul	Tabu	
_____	2048	2048	2048	**neighbors**
_____	_____	_____	20	**tabu-list**
_____	nXn	_____	_____	**probMat**
_____	nXn	_____	_____	**PhermonMat**
_____	n	_____	_____	**ProbVector**
2048	_____	_____	_____	**population/next**

וסים טנוס: 207866328
פארוק כריים: 314656869

ומשותף ביניהם המקום של קלאס הבעיה שמצוין בתמונה:

```
class CVRP:

    def __init__(self, distanceMatrix, depot, cities, capacity, size):
        self.distanceMatrix = distanceMatrix
        self.cities = cities
        self.depot = depot
        self.capacity = capacity
        self.size = size
        self.best = []
        self.bestFitness = 0
```

distanceMat = nXn
cities = n

<u>וכל השאר O(1)</u>

10. עבור כל אלגוריתם וכל בעיה יש לשרטט גרפים:

10.1 של ערך ההיוריסטיקה הטוב ביותר בכל איטרציה של האלגוריתם

10.2 של איכות הפתרון כפונקציה של מספר האיטרציות

<u>10.1:</u>

נבני טבלה מסכמת כל האלגוריתמים. (בהסתמכות על הגרפים שבסעיף 7)

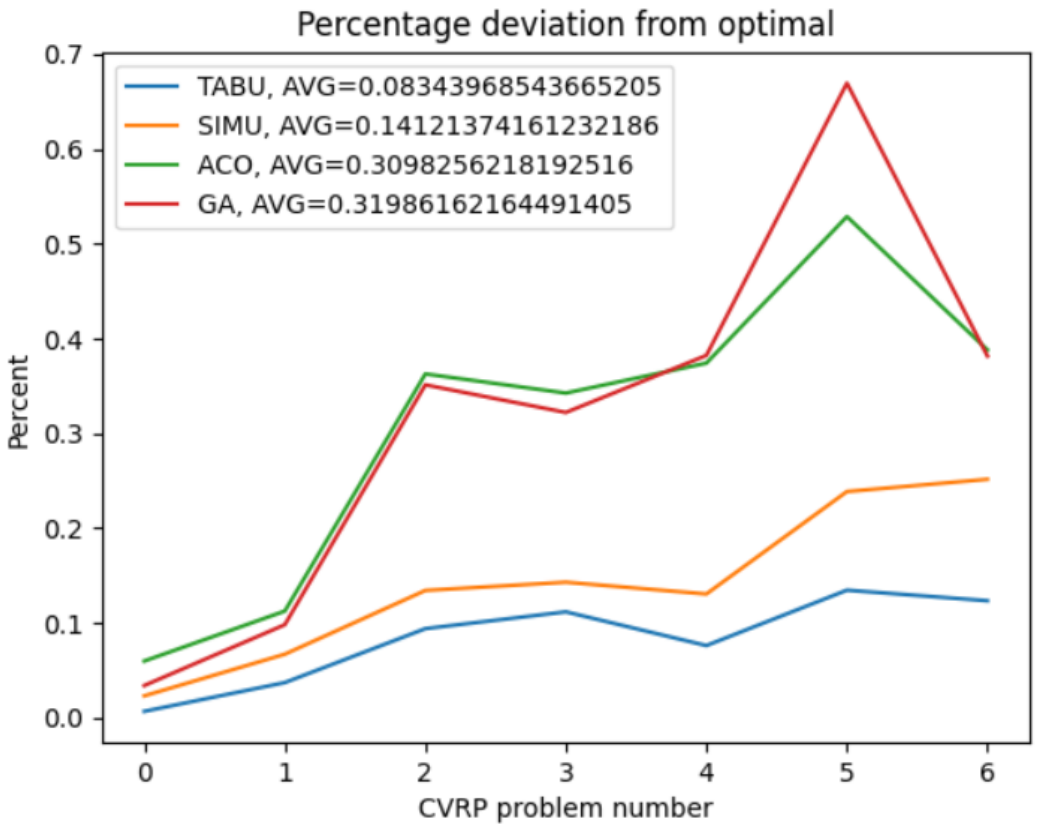
	1	2	3	4	5	6	7
OPTIMAL	375	835	521	832	735	817	1077
TABU	375	842	532	865	761	842	1121
SIMU	375	847	563	904	799	961	1235
ACO	378	877	612	924	828	940	1253
GA	379	881	614	916	799	962	1200

וסים טנוס: 207866328
פארוק כריים: 314656869

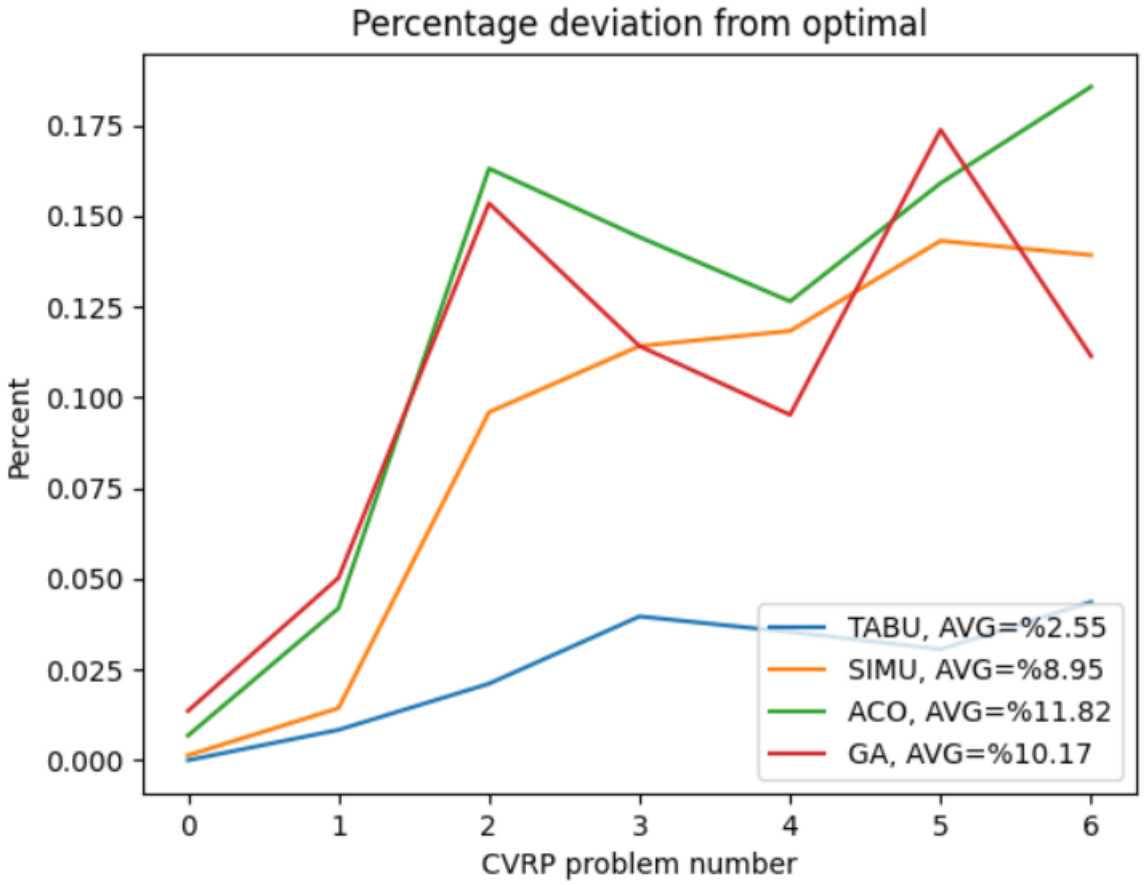
10.2:

שוב בהסתמכות על הגרפים שבסעיף 7 נבנה גרף המסכם האיכות של ממוצע ה-FITNESS שקיבלנו מ-10 האיטרציות עבור כל אלגוריתם.
הגרף מכיל את ממוצע האיכות של כל האלגוריתמים עבור כל הבעיות.

יש שני גרפים: הגרף הראשון מסכם 7 הגרפים הראשונים שבסעיף 7.
הגרף השניי מסכם 7 הגרפים האחרונים (הטובים)



וסים טנוס: 207866328
פארוק כריים: 314656869



11. נתחו באופן מילולי כל אלגוריתם עפ"י הגרפים שיצרתם, והסבירו עבור אילו תכונות קלט לדעתכם אלגוריתמים אלה יעילים.

כמו שרואים בגרפים לעיל עבוד בעיות קטנות (מספר ערים קטן) האלגוריתמים עובדים בצורה טובה מאוד ונותנים פתרונות תת אופטימליים קרובים לאופטימלי. עבוד TABU מבחינתינו הוא הטוב ביותר כי גם עבור בעיות גדולות הוא לא מתרחק הרבה מאופטימלי.
עבור SIMULATED גם כן אלגוריתמם מעולה מתקרב הרבה לאופטימלי אבל עבור בעיות גדולות יש יותר סיכוי לסטות.
עבור GA +ACO אנחנו יכולים לראות שהוא כמעט כמו SIMULATED - עובד טוב בבעיות קטנות, ועבור בעיות גדולות לא יתקרבו הרבה מאופטימלי.
הערה מאוד חשובה: לפי הגרפים לעיל אנחנו יכולים להשיג המון על השפעת הקונפיגורציה ההתחלתית על הפתרון שמתקבל. כך עבוד קונפיגוראציה רנדומליים יש פחות סיכוי להתקרב לאופטימום.

וסים טנוס: 207866328
פארוק כריים: 314656869

TABU	**: 8%**	⇒	**2.5%**
SIMULATED ANNEALING	**:14%**	⇒	**8.9%**
GA	**:30%**	⇒	**11.8%**
ACO	**:31%**	⇒	**10.1%**

עבור השיפור ב-ACO היה בג שם בחישוב הפירמונים, מה שאומר שמאוד חשוב עניין הפירמונים עבור כל נמלה כך שאם טעינו בחישוביהם נתרחק הרבה מאופטימום (לא נתקרב).

12. נסו לשערך את הסקלביליות של כ"א מהאלגוריתמים: קרי את הקשר בין גודל הבעיה לזמן הריצה של כ"א מהאלגוריתמים

נסתמך במסקנות שלנו על כל הגרפים שנמצאים בדוח, ועל גרף האיכות (מסעיף 10.2) ספציפי.

לגבי הזמן: כמו שאמרנו מקודם, הגבלנו את האלגוריתם במספר איטרציות קבוע, וכמובן שאם המספר גדל, איכות האלגוריתם תגדל.

ה-TABU:

אנחנו יכולים לסכם את איכום הפתרון שנותן האלגוריתם עבור כל הבעיות באופן כללי כטוב כי הוא סוטה בממוצע לא מעל 2% שזה אומר שהוא מצויין. ולפי גרף האיכום אנחנו יכולים להשיק שבכל פעם שמספר הערים גדל בבעיה, אחוז הסטייה יגדל קצת.

וסים טנוס: 207866328
פארוק כריים: 314656869

ה-SIMU:

אנחנו יכולים לסכם את איכום הפתרון שנותן האלגוריתם עבור כל הבעיות באופן כללי כטוב יחסית לשאר האלגוריתמים כי הוא סוטה בממוצע 8% שזה אומר שהוא טוב. ולפי גרף האיכום אנחנו יכולים להשיק שבכל פעם שמספר הערים גדל בבעיה, אחוז הסטייה יגדל.

ה-:ACO:

אנחנו יכולים לסכם את איכום הפתרון שנותן האלגוריתם עבור כל הבעיות באופן כללי כטוב במידה מסוימת כי הוא סוטה בממוצע 11%. ולפי גרף האיכום אנחנו יכולים להשיק שהאלגוריתם עובד מצויין עבור בעיות קטנות. בכל פעם שמספר הערים גדל בבעיה, יש סיכוי יותר לא להתקרב מהאופטימל.

ה-GA:

אנחנו יכולים לסכם את איכום הפתרון שנותן האלגוריתם עבור כל הבעיות באופן כללי כטוב כי הוא סוטה בממוצע 10%. ולפי גרף האיכום אנחנו יכולים להשיק שהאלגוריתם עובד מצויין עבור בעיות קטנות. בכל פעם שמספר הערים גדל בבעיה, יש סיכוי יותר גדול לא להתקרב מהאופטימל. מכיוון שהוא אלגוריתם הסתברותי לבחירת ההורים שמהם נוצר בנים, גם יש סיכוי לא להתקרב לפתרון כמו שצריך, כלומר נקביל נדנודים.

סיכום האלגוריתם מהטוב לפחות טוב:

TABU
SIMULATED ANNEALING
GA
ACO