AIP

HW 08 – REPORT

소속 : 정보컴퓨터공학부

학번 : 202255535

이름 : 김진우

1. 서론

이번 과제에서는 stochastic Hill Climbing, random-restart, simulated annealing 알고리즘을 추가한 프로그램을 설계해야 한다. 우선 Stochastic Hill climbing은 evaluate값에 따른 확률을 배정한 후 이 확률로 neighbor를 뽑아 이전보다 좋아지면 새로운 값으로 solution을 대체하고 이 과정을 더 이상 좋아지지 않을 때까지 반복하는 것이다. Random-Restart는 말 그대로 알고리즘을 처음부터 다시 실행하는 것으로 시작점을 바꾸어 실행할 때 가장 좋은 것을 선택하는 알고리즘이다. Simulated-Annealing은 기존의 hill-climbing 방법과 유사하지만 차이점이 존재하는 알고리즘인데 바로 "안 좋아지는 방향"도 허용한다는 점이다. 기존의 알고리즘은 좋아지는 방향만 허용하기 때문에 local minimum(max)에 갇혀 빠져나오지 못할 가능성이 있는데 안 좋아지는 방향도 허용을 하면 더 좋은 값을 찾을 수 있게 된다. 또한 이번 과제에서는 exp.txt파일 내의 값만 수정하여 main을 실행시키기 때문에 이에 맞춰 main.py를 수정해야 하고 plot.py를 작성하여 그래프를 통해 성능 비교까지 해봐야 한다.

2. 본론

2.1 main.py

Main.py에서는 exp.txt에서 정보를 읽어와 가장 좋은 결과를 출력해야한다. Main.py는 main-skeleton.py에서 비어있는 createProblem/createOptimizer를 구현하여 완성하면된다.

2.1.1 createProblems(parameters)

이 함수는 parametes에 저정되어있는 pType을 읽어와서 pType이 1이면 Numeric()을 2이면 Tsp() 문제가 되게하고 setVariables를 통해 설정을 한 뒤 p를 return한다.

```
def createProblem(parameters): ###
    # Create a problem instance (a class object) 'p' of the type as
    # specified by 'pType', set the class variables, and return 'p'.
    if (parameters['pType'] == 1):
        p = Numeric()
    elif (parameters['pType'] == 2):
        p = Tsp()
    p.setVariables(parameters)
    return p
```

2.1.2 createOptimizer(parameters)

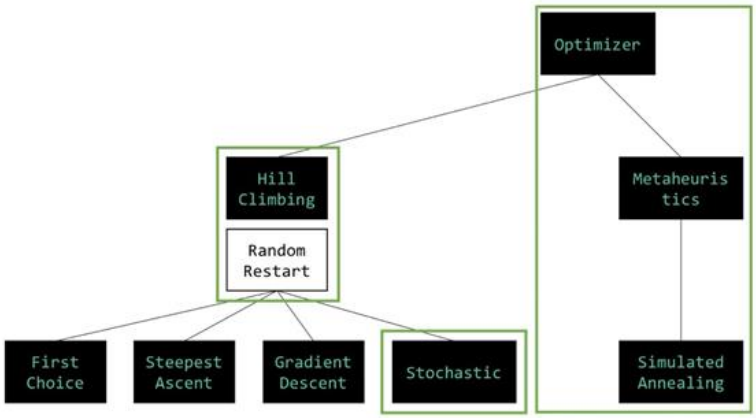
이 함수는 aType을 읽어와 aType에 맞는 알고리즘을 선택하고 setVariables를 통해 필요

한 변수들을 생성하고 이를 return한다.

```
def createOptimizer(parameters): ###
    # Create an optimizer instance (a class object) 'alg' of the type
    # as specified by 'aType', set the class variables, and return 'alg'.
    optimizers = {1: 'SteepestAscent()',
                  2: 'FirstChoice()',
                  3: 'Stochastic()',
                  4: 'GradientDescent()',
                  5: 'SimulatedAnnealing()'
                  }
    aType = parameters['aType']
    alg = eval(optimizers[aType])
    alg.setVariables(parameters)
    return alg
```

2.2 optimizer.py

여기서는 새롭게 Optimizer class를 생성하여 HillClimibing과 Metaheuristic이 이를 상속하게 한다. 그 다음 Hillclimbing을 First-choice, Steepest-Ascent, Gradient Descent, Stochastic이 상속하도록 하고 Metaheuristic은 Simulated Annealing이 상속하도록 class화 한다. (아래의 그림 참고)



2.2.1. Optimizer(Setup)

Setup class를 super class로 가지며 numExp 변수를 저장한다. setVariables 메소드에서는 Setup의 setVariables로 변수를 설정하고 numExp도 추가로 설정해준다. numExp를 위한 accessor 메소드도 구현을 하고 numExp를 출력 양식에 맞춰 터미널에 출력해주는 displayNumExp 메소드도 같이 구현을 해준다.

```
# Optimizer class가 새로 생성됨
class Optimizer(Setup):
    def __init__(self):
        Setup.__init__(self)
        self._numExp = 0

    def setVariables(self, parameters):
        Setup.setVariables(self, parameters)
        self._numExp = parameters['numExp']

    def getNumExp(self):
        return self._numExp

    def displayNumExp(self):
        print()
        print("Number of experiments: {}".format(self._numExp))
```

2.2.2 HillClimbing(Optimizer)

Setup이 아닌 새로운 Optimizer class를 상속받는다.

변한 부분만 구체적으로 하나씩 살펴보자면

- __init__

이 메소드는 Setup이 아닌 새로운 Optimizer class를 상속받고, limitStuck와 numRestart라는 변수를 저장한다

```
def __init__(self):
    Optimizer.__init__(self)
    #self._pType = 0
    self._limitStuck = 1000 #변하는 부분
    self._numRestart = 0 #변하는 부분
```

- setVariables(parameters)

각종 변수들을 설정하기 위한다. 이때 부모 class의 setVariables를 사용하고 limitStuck과 numRestart에 대해서도 설정을 해준다.

```
def setVariables(self, parameters):
    Optimizer.setVariables(self, parameters)
    self._limitStuck = parameters['limitStuck']
    self._numRestart = parameters['numRestart']
```

- randomRestart(p)

이번 과제에서는 Hill-climbing에 대해서 randomRestart를 적용시키기 때문에 이를 위한 메소드를 작성한다. 이 코드는 실습에서 진행하였기 때문에 설명은 생략하도록 하겠다.

```python
def randomRestart(self, p):
    p.setNumEval(0)
    i = 1
    self.run(p)
    bestSolution = p.getSolution()
    bestMinimum = p.getValue()
    numEval = p.getNumEval()
    while i < self._numRestart:
        self.run(p)
        newSolution = p.getSolution()
        newMinimum = p.getValue()
        numEval += p.getNumEval()
        if newMinimum < bestMinimum:
            bestSolution = newSolution
            bestMinimum = newMinimum
        i += 1
    p.storeResult(bestSolution, bestMinimum)
```

- displaySetting

출력 양식에 맞게 출력되도록 displaySetting 메소드를 작성한다. 자세한 설명은 아래 사진에 첨부된 코드와 주석을 참고하길 바란다.

출력 형식은 모두 과제와 함께 첨부된 pdf와 실습 ppt를 참고하였다.

```python
def displaySetting(self):
    if self._numRestart > 1:
        print("Number of random restarts: {}".format(self._numRestart))
        print()
    #numeric이고 gradient descent아니면 delta값 출력
    if(self._pType == 1 and self._aType != 4):
        print("Mutation step size:", self._delta)
    # first choice or stochastic이면 limitStuck 출력
    if(self._aType == 2 or self._aType == 3):
        print("Max evaluations with no improvement: {0:,} iterations".format(self._limitStuck))
    #numeric이고 Gradient Descent일경우에는 alpha값과 dx값 출력
    elif(self._pType == 1 and self._aType == 4):
        print("Update rate:", self._alpha)
        print("Increment for calculating derivatives:", self._dx)
```

2.2.3. FirstChoice, SteepestAscent, GradientDescent(Hillcliming)

알고리즘 수행 과정에서 File I/O를 사용해 매 iteration마다 결과를 저장해야하기 위한 코드를 run 메소드에 추가한다. 3개의 class모두 같은 흐름이기 때문에 묶어서 설명하였다.

```
def run(self, p):
    # first-choice.py에 정의했던 firstchoice 함수를 활용해서 구현
    current = p.randomInit()
    valueC = p.evaluate(current)
    f = open("FirstChoice.txt", 'w')   # 파일 생성
    i = 0
    while i < self._limitStuck:
        successor = p.randomMutant(current)
        valueS = p.evaluate(successor)
        f.write(str(valueC) + '\n') # 파일에 쓰기 -> 매 iteration마다 결과를 저장해야하기때문
        if valueS < valueC:
            current = successor
            valueC = valueS
            i = 0
        else:
            i += 1
    f.close()
    p.storeResult(current, valueC)
```

```
def run(self, p):
    current = p.randomInit() # 'current' is a list of values
    valueC = p.evaluate(current)
    f = open("SteepestAscent.txt", 'w')
    while True:
        neighbors = p.mutants(current)
        successor, valueS = self.bestOf(neighbors, p)
        f.write(str(valueC) + '\n')
        if valueS >= valueC:
            break
        else:
            current = successor
            valueC = valueS
    p.storeResult(current, valueC)
    f.close()
```

```
def run(self,p):
    currentP = p.randomInit()  # Current point
    valueC = p.evaluate(currentP)
    f = open("GradientDescent.txt", 'w')
    while True:
        nextP = p.takeStep(currentP, valueC)
        valueN = p.evaluate(nextP)
        f.write(str(valueC) + '\n')
        if valueN >= valueC:
            break
        else:
            currentP = nextP
            valueC = valueN
    p.storeResult(currentP, valueC)
    f.close()
```

2.2.4 Stochastic(Hillclimbing)

Hillclimbing class를 상속하며 displaySetting, run, stochasticBest method를 정의한다. Stochastic알고리즘은 Steepest Ascent 알고리즘과 흐름이 유사하다는 점을 활용해서 run 메소드를 구현하고 이곳에서도 앞에서와 마찬가지로 매 iteration마다 결과를 저장해야하기 위한 코드를 함께 구현한다. stochasticBest 메소드는 함께 제공된 some-useful-codes.py를 참고하였다.

```
class Stochastic(HillClimbing):
    def __init__(self):
        pass
    def displaySetting(self):
        print()
        print("Search algorithm: Stochastic Hill Climbing")
        print()
        HillClimbing().displaySetting()
    def run(self, p):
        # hint; Stochastic알고리즘은 Steepest Ascent 알고리즘과 흐름이 유사함
        current = p.randomInit()
        valueC = p.evaluate(current)
        f = open("Stochastic.txt", 'w') # 파일 생성
        i = 0
        while i < self._limitStuck: # limitStuck만큼 반복
            neighbors = p.mutants(current) # 현재 위치에서의 이웃들을 찾음
            successor, valueS = self.stochasticBest(neighbors, p) # 이웃을 확률적으로 선택
            f.write(str(valueC) + '\n')
            if valueS < valueC: # 선택된 이웃이 더 좋은 경우(값이 작은 경우)
                current = successor
                valueC = valueS
                i = 0
            else:
                i += 1
        p.storeResult(current, valueC)
        f.close()
    # some useful codes.py의 stochasticBest 함수를 활용
    def stochasticBest(self, neighbors, p):
        # Smaller valuse are better in the following list
        valuesForMin = [p.evaluate(indiv) for indiv in neighbors]
        largeValue = max(valuesForMin) + 1
        valuesForMax = [largeValue - val for val in valuesForMin]
        # Now, larger values are better
        total = sum(valuesForMax)
        randValue = random.uniform(0, total)
        s = valuesForMax[0]
        for i in range(len(valuesForMax)):
            if randValue <= s: # The one with index i is chosen
                break
            else:
                s += valuesForMax[i+1]
        return neighbors[i], valuesForMin[i]
```

2.2.5. MetaHeuristics(Optimizer)

Optimzer를 상속받는 클래스로 앞선 클래스들과 마찬가지로 setVariables를 통해 변수를 저장하고 출력 형식에 맞게 출력되도록 displaySetting를 설정해준다.

```
class Metaheuristic(Optimizer):
    def __init__(self):
        Optimizer.__init__(self)

    def setVariables(self, parameters):
        Optimizer.setVariables(self, parameters)
        self._numExp = parameters['numExp']
        self._numSample = 10
        self._limitEval = parameters['limitEval']

    def displaySetting(self):
        if self._pType == 1: # numeric일 경우에만 dx값 출력되도록
            print("Mutation step size:", self._delta)
```

2.2.6. SimulatedAnnealing(MetaHeuristics)

이 클래스는 MetaHeuristics class 를 상속받으며 numSample 변수를 가지며, displaySetting, run, initTemp, tSchedule method 를 가진다. 이때 initTemp, tSchedule method 는 some-useful-codes.py 에서 주어졌기 때문에 설명을 하지 않고 넘어가도록 하겠다.

```
class SimulatedAnnealing(Metaheuristic):
    def __init__(self):
        Metaheuristic.__init__(self)
        self._numSample = 100
        self._limitEval = 10000
        self._whenBestFound = 0

    def getWhenBestFound(self):
        return self._whenBestFound

    def displaySetting(self):
        print()
        print("Search algorithm: Simulated Annealing")
        print()
        print("Number of evaluations until termination: {0:,}".format(self._limitEval))
        super().displaySetting()
```

Run 메소드에 대해서 자세히 설명하자면 tSchedule을 통해 t를 업데이트하고 현재 상태에 대한 randomMutant를 생성한다. 그 다음 p.evaluate를 통해 mutant값을 평가하고 mutant의 값이 좋다면 무조건 업데이트를 하고 그렇지 않은 경우에는 확률에 따라 업데이트하게 된다. 즉 값이 더 나쁘더라도 확률적으로 변경될 수 있다는 것을 의미한다.

그 다음 update된 값이 best값보다 좋으면 best update를 진행하도록 코드를 구현해야한다.

```
def run(self, p):
    current = p.randomInit()
    valueC = p.evaluate(current)
    best, valueBest = current, valueC
    self.whenBestFound = i = 1
    t = self.initTemp(p)
    f = open("SimulatedAnnealing.txt", 'w')
    while True:
        t = self.tSchedule(t) # tSchedule을 통해 t값을 업데이트
        f.write(str(valueC) + '\n')
        if(t == 0 or i==self._limitEval): #temperature가 0이 되거나 limitEval에 도달하면 종료
            break
        neighbor = p.randomMutant(current) # 현재 상태에 대한 randomMutant 생성
        valueN = p.evaluate(neighbor) # mutant 평가

        dE = valueN - valueC
        if(valueN - valueC < 0): # 값이 더 좋다면 업데이트
            current = neighbor
            valueC = valueN
        # 값이 안좋다면 확률적  update
        #이때 확률은 tempture에 따라 결정
        elif random.uniform(0,1) < math.exp(-dE/t):
            current = neighbor
            valueC = valueN
        # 업데이트 된 값이 best보다 좋다면 best 업데이트
        if valueC < valueBest:
            (best, valueBest) = (current, valueC)
            whenBestFound = i

        i+=1 #iteration 증가

    self._whenBestFound = whenBestFound
    p.storeResult(best, valueBest)
    f.close()
```

2.3. problem.py

2.3.1. Problem class

Problem class는 setup class를 상속한다. 이 클래스는 파일의 이름, bestSolution 등과 같은 변수를 저장한다. 그리고 setVariable 메소드를 통해 파일명과 Setup에서 정의하는 변수들을 설정한다. Solution과 value, numEval에 접근하기 위한 accessor 메소드로 구현하며 numEval의 값을 초기화 시켜주기 위한 setNumEval이라는 메소드도 구현한다. 마지막으로 실험의 결과를 저장할 storeExpResult라는 메소드까지 구현하면 된다.
또한 공통적으로 마지막에 출력되는 evaluation의 개수를 출력하는 것을 sumOfNumEval을 사용해 출력 형식에 맞게 출력되도록 코드를 작성한다.

이때 변수를 저장하는 이유는 random-restart방식을 사용하기 때문이다.

```
# 결과 저장
def storeExpResult(self, results):
    self._bestSolution  = results[0]
    self._bestMinimum   = results[1]
    self._avgMinimum    = results[2]
    self._avgNumEval    = results[3]
    self._sumOfNumEval  = results[4]
    self._avgWhen       = results[5]
```

```
class Problem(Setup):
    def __init__(self):
        Setup.__init__(self)
        self._solution = []
        self._value = 0
        self._numEval = 0

        # 8번째 과제에서 추가된 부분 -> 제공된 PPT 대로 추가함
        self._pfileName = ''
        self._bestSolution = []
        self._bestMinimum = 0
        self._avgMinimum = 0
        self._avgNumEval = 0
        self._sumOfNumEval = 0
        self._avgWhen = 0

    # 파일명과 Setup에 정의된 변수들을 받아서 self._pfileName에 assign
    def setVariables(self, parameters):
        Setup.setVariables(self, parameters)
        self._pfileName = parameters['pFileName']

    # 8번째 과제에서 추가된 부분 : accessor method들 추가
    def getSolution(self):
        return self._solution
    def getValue(self):
        return self._value
    def getNumEval(self):
        return self._numEval

    def setNumEval(self, numEval):
        self._numEval = numEval
```

```
def report(self):
    print()
    print("Total number of evaluations: {0:,}".format(self._sumOfNumEval))
```

2.3.2. Numeric class

변경된 부분에 대해서만 설명해보면

- setVariables(parameters)

이전과는 다르게 parameters라는 것을 파라미터로 추가로 받게 구현한다. 그리고 Problem class의 setVariables를 호출하여 필요한 변수들을 정의한다.

```
def setVariables(self, parameters): # parameters를 파라미터로 받아서 변수들을 설정
    Problem.setVariables(self, parameters) # 부모 클래스의 setVariables 호출
    filename = parameters['pFileName'] # 파일명을 filename에 저장
    infile = open(filename, "r")
    self._expression = infile.readline()
    varNames, low, up = [], [], [] # 각각 변수 이름, 최소값, 최대값을 저장할 리스트
    line = infile.readline()

    while line != "":
        data = line.split(",")
        varNames.append(data[0])
        low.append(eval(data[1]))
        up.append(eval(data[2]))
        line = infile.readline()
    infile.close()
    self._domain = [varNames, low, up]
```

- report

출력 양식에 맞게 출력되도록 코드를 설정한다. 이때 Numeric problem을 풀 때 출력되는 형식은 첨부된 pdf의 출력 형식을 참고하였다.

```
def report(self):   # 출력양식에 맞게 출력되게 코드 수정
    print()
    print("Average objective value: {0:.3f}".format(self._avgMinimum))
    print("Average number of evaluations: {0:,}".format(self._avgNumEval))
    print()
    print("Best solution found:")
    print(self.coordinate())
    print("Best value: {0:,.3f}".format(self._bestMinimum))
    Problem.report(self)
```

- coordinate

이 메소드에서는 이전에는 그냥 solution을 return했다면 이번 과제에서는 bestSolution을 return하도록 코드를 수정하였다.

```
def coordinate(self):
    c = [round(value,3) for value in self._bestSolution] # bestSolution이 출력되게 수정
    return tuple(c)
```

2.3.3. Tsp class

Numeric class와 흐름이 유사하게 이전 코드로부터 약간의 추가 및 수정을 해주면 된다.

- setVariables(parameters)

```
def setVariables(self,parameters): # parameters를 파라미터로 받아서 변수들을 설정
    ## Read in a TSP (# of cities, locatioins) from a file.
    ## Then, create a problem instance and return it.
    fileName = parameters['pFileName']
    Problem.setVariables(self, parameters) # 부모 클래스의 setVariables 호출
    infile = open(fileName, 'r')
    # First line is number of cities
    self._numCities = int(infile.readline())
    locations = []
    line = infile.readline()  # The rest of the lines are locations
    while line != '':
        locations.append(eval(line)) # Make a tuple and append
        line = infile.readline()
    infile.close()
    self._locations = locations
    self._distanceTable = self.calcDistanceTable()
```

- report

출력 형식에 맞게 출력되도록 하였는데 TSP의 경우는 조교님과 함께 진행한 실습 ppt의 출력 형식을 참고하였다.

```
def report(self):   #출력 형식에 맞게 출력되게 코드 수정
    print()
    print("Average tour cost: {}".format(round(self._avgMinimum)))
    print("Average number of evaluations: {0:,}".format(self._avgNumEval))
    print()
    print("Best tour found:")
    self.tenPerRow()
    print("Best tour cost: {0:,}".format(round(self._bestMinimum)))
    Problem.report(self)
```

-tenPerRow

Numeric의 coordinate와 마찬가지로 이전까지 리턴했던 solution값 대신에 bestSolution을 리턴하도록 코드를 수정한다.

```
def tenPerRow(self):
    solution = self._bestSolution # bestSolution을 저장하는 것으로 코드 수정
    for i in range(len(solution)):
        print("{0:>5}".format(solution[i]), end='')
        if i % 10 == 9: # 10개씩 출력 -> 줄바꿈
            print()
```

2.4 setup.py

Setup class에서는 aType이라는 것이 추가되고 aType을 get하기 위해 getAType도 추가되게 된다. 또한 변수를 Parameters에 저장된 값으로 저장하기 위해 setVariables도 추가해 준다.

```
class Setup:
    def __init__(self):
        self._pType = 0
        self._aType = 0 # 추가된 부분
        self._delta = 0
        self._alpha = 0
        self._dx = 0

    def setVariables(self, parameters):
        self._pType = parameters['pType']
        self._aType = parameters['aType']
        self._delta = parameters['delta']
        self._alpha = parameters['alpha']
        self._dx = parameters['dx']

    def getAType(self):
        return self._aType
```

2.5 plot.py

Run method를 통해 생성된 txt파일을 이용해 변화를 시각화하기위한 코드이다. Firstchoice와 SimulatedAnnealing에서 numEval에 따른 Current Value를 시각화하는 것이 과제였기 때문에 이 2개이 파일로 한정하여 코드를 작성하였다.

```
import matplotlib.pyplot as plt

def main():
    myfile = open("FirstChoice.txt", "r")
    firstList = []
    while True:
        val = myfile.readline().rstrip() # 파일을 읽어서 한줄씩 읽어오기
        if val == "": # 파일이 끝나면
            break
        firstList.append(eval(val)) # 파일에 있는 값을 리스트에 추가
    myfile.close()

    myfile = open("SimulatedAnnealing.txt", "r")
    annealList = []
    while True:
        val = myfile.readline().rstrip()
        if val == "":
            break
        annealList.append(eval(val))
    myfile.close()

    plt.plot(range(len(firstList)), firstList)
    plt.plot(range(len(annealList)), annealList)
    plt.xlabel("Number of Evaluations")
    plt.ylabel("Tour Cost")
    plt.title("Search Performance (TSP-100)")
    plt.legend(["First-Choice HC", "Simulated Annealing"])
    plt.show()

main()
```

2.6 실행 결과

- Stochastic – Numeric : Ackley.txt

```
Enter the file name of experimental setting: exp.txt

Objective function:
20 + math.e - 20 * math.exp(-(1/5) * math.sqrt((1/5) * (x1 ** 2 + x2 ** 2 + x3 ** 2 + x4 ** 2 + x5 ** 2))) - math.exp((1/5) * (math.cos(2 * math.p
i * x1) + math.cos(2 * math.pi * x2) +math.cos(2 * math.pi * x3) +math.cos(2 * math.pi * x4) +math.cos(2 * math.pi * x5)))

Search space:
 x1: (-30, 30)
 x2: (-30, 30)
 x3: (-30, 30)
 x4: (-30, 30)
 x5: (-30, 30)

Number of experiments: 5

Search algorithm: Stochastic Hill Climbing
Number of random restarts: 10

Mutation step size: 0.01
Max evaluations with no improvement: 100 iterations

Average objective value: 17.207
Average number of evaluations: 51,142

Best solution found:
(0.004, -9.983, -3.997, 8.98, 1.998)
Best value: 14.367

Total number of evaluations: 255,710
```

- Stochastic – TSP : tsp30.txt

```
Enter the file name of experimental setting: exp.txt

Number of cities: 30
City locations:
     (8, 31)    (54, 97)    (50, 50)    (65, 16)    (70, 47)
   (25, 100)    (55, 74)    (77, 87)     (6, 46)    (70, 78)
    (13, 38)   (100, 32)    (26, 35)    (55, 16)    (26, 77)
    (17, 67)    (40, 36)    (38, 27)     (33, 2)     (48, 9)
    (62, 20)    (17, 92)     (30, 2)    (80, 75)    (32, 36)
    (43, 79)    (57, 49)    (18, 24)    (96, 76)    (81, 39)

Number of experiments: 5

Search algorithm: Stochastic Hill Climbing
Number of random restarts: 10

Max evaluations with no improvement: 100 iterations

Average tour cost: 472
Average number of evaluations: 190,480

Best tour found:
   11   29    4   26    2    3   20   13   19   18
   22   17   16   24   12   27    0   10    8   15
   14   21    5    1   25    6    9    7   23   28
Best tour cost: 457

Total number of evaluations: 952,401
```

- Simulated Annealing – Numeric : Ackley.txt

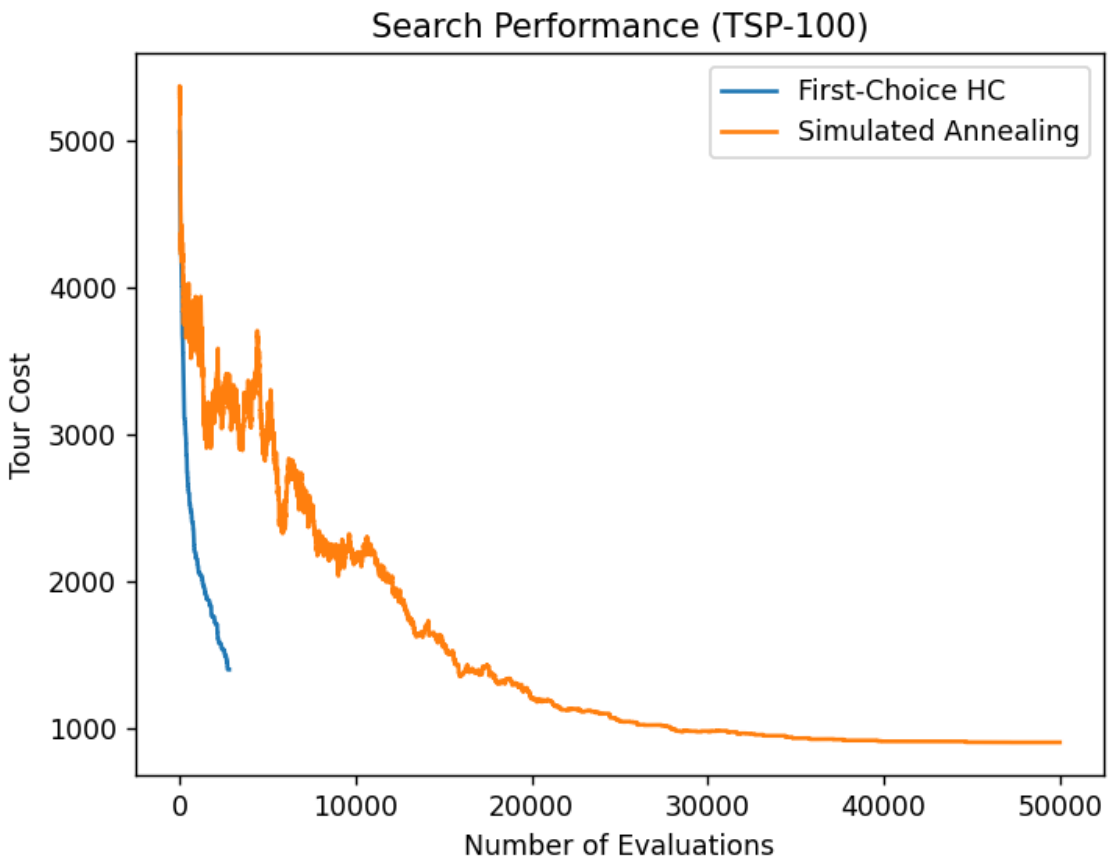
```
Enter the file name of experimental setting: exp.txt

Objective function:
20 + math.e - 20 * math.exp(-(1/5) * math.sqrt((1/5) * (x1 ** 2 + x2 ** 2 + x3 ** 2 + x4 ** 2 + x5 ** 2))) - math.exp((1/5) * (math.cos(2 * math.pi * x1) + math.cos(2 * math.pi *
x2) +math.cos(2 * math.pi * x3) +math.cos(2 * math.pi * x4) +math.cos(2 * math.pi * x5)))

Search space:
 x1: (-30, 30)
 x2: (-30, 30)
 x3: (-30, 30)
 x4: (-30, 30)
 x5: (-30, 30)

Number of experiments: 5

Search algorithm: Simulated Annealing

Number of evaluations until termination: 1,000
Mutation step size: 0.01

Average objective value: 19.140
Average number of evaluations: 3,060

Best solution found:
(-1.999, 4.997, -11.998, -18.996, -8.988)
Best value: 17.824

Total number of evaluations: 15,300
```

- Simulated Annealing – TSP : tsp30.txt

```
Number of cities: 30
City locations:
     (8, 31)    (54, 97)    (50, 50)    (65, 16)    (70, 47)
   (25, 100)    (55, 74)    (77, 87)     (6, 46)    (70, 78)
    (13, 38)   (100, 32)    (26, 35)    (55, 16)    (26, 77)
    (17, 67)    (40, 36)    (38, 27)     (33, 2)     (48, 9)
    (62, 20)    (17, 92)     (30, 2)    (80, 75)    (32, 36)
    (43, 79)    (57, 49)    (18, 24)    (96, 76)    (81, 39)

Number of experiments: 5

Search algorithm: Simulated Annealing

Number of evaluations until termination: 1,000

Average tour cost: 938
Average number of evaluations: 3,060

Best tour found:
   14   13    3   20   19   22   18    8   10   27
   17    4    7   23    6    9   12   16    1   25
   24    0    2   26   11   29   28   21    5   15
Best tour cost: 878

Total number of evaluations: 15,300
```

- Plot.py

Tsp100.txt사용(limitStuck = 100, numeval = 50000 -> 이 값이 exp.txt에 적힌 초기값임)



3. 결론

First-Choice HC의 경우 초기 단계에서 비용이 급격히 감소하는 것을 볼 수 있다. 그러나 특정 시점 이후 더 이상의 개선이 이루어지지 않는다는 것을 확인할 수 있다. 이는 Hill Climbing 알고리즘의 특성 상 더 나은 해를 찾지 못하고 local minimum에 빠진 상태라고 해석할 수 있다.

Simulated Annealing의 경우 초기에는 불안정한 경로 비용 감소를 보이지만, 시간이 지남에 따라 비용이 점진적으로 감소하는 것을 확인할 수 있다. 마지막에는 First-Choice HC보다 더 낮은 비용으로 도달하게 된다. 즉 Simulated Annealing은 탐색이 진행될수록 점진적으로 최적해에 가까워지는 특성이 있다고 할 수 있다.

정리하자면 First-Choice HC는 빠르게 해를 찾지만, 탐색이 제한적이고 local minimum에 빠질 확률이 낮은 cost의 경로를 찾지 못한다. 반면 Simulated Annealing은 초기에는 불안정하고 오히려 cost가 높아지는 방향으로 이동하기도 하지만, 더 많은 평가 횟수를 통해 꾸준히 비용을 낮추며 최적화된다. 결과적으로 First-choice HC보다 더 낮은 cost에 최종적으로 도달하게 됨을 확인할 수 있다.