Faculty of Computer & Information Sciences
Ain Shams University
Subject: CSC 343 Artificial Intelligence
Year: (3rd year)undergraduate
Academic year: 2nd term 2019-2020

Research Topic (3)

Title: Evolutionary Computation, Classification and Search

1. Introduction

1.1A* algorithm:

It's a smart searching algorithm we also could say that it has brain, there are other algorithms similar to A* for example Dijkstra algorithm, Dijkstra algorithm is special case of A* algorithm, where heuristic value is zero for all node.
A* algorithm is Used in many games and web-based maps for path finding and graph traversals, Aim : We want to reach the target cell (if possible) from the starting cell as quickly as possible

1.2K-NN algorithm:

This algorithm is used to solve the classification model problems. K-nearest neighbor or K-NN algorithm basically creates an imaginary boundary to classify the data. When new data points come in, the algorithm will try to predict that to the nearest of the boundary line.

Therefore, larger k value means smother curves of separation resulting in less complex models. Whereas, smaller k value tends to overfit the data and resulting in complex models.

1.3Genetic algorithm:

A **genetic algorithm** is a search heuristic that is inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation

2. The algorithms

2.1. A*

2.1.1. The main steps of the algorithm

Step 1: Picks start node and select its successors

Step 2: Picks node from the successors with lowest f(n)=g(n)+h(n) which is g(n) equal to the total cost from start node to current node n, h(n) is heuristic function equal to the cost from current node n to target node.

Step 3: Makes the new node the start node and repeat step 1 till reaching to the target node.

2.1.2. The implementation of A* algorithm

```python
class SearchAlgorithms:
    path = []  # Represents the correct path from start node to the goal node.
    fullPath = []  # Represents all visited nodes from the start node to the goal node.
    totalCost = None
    #my attribute
    mazeMap = {}
    costMap = {}
    start = None
    end = None
    iid=0
    def __init__(self, mazeStr, edgeCost):
        i = 0
        j = 0
        ii = 0
        while (ii < len(mazeStr)):
            if mazeStr[ii] == ' ':
                j += 1
                i = 0
                ii += 1
                continue
            if mazeStr[ii] == 'S':
                self.start = (j,i)
            elif mazeStr[ii] == 'E':
                self.end = (j,i)
            if mazeStr[ii] != ',':
                self.mazeMap[(j,i)] = mazeStr[ii]
                i += 1
            ii += 1
        iii = 0
        l = 0
        m = 0
        while (iii < len(edgeCost)):
            self.costMap[(m, l)] = edgeCost[iii]
            l += 1
            iii += 1
            if (iii % 7 == 0):
                l = 0
                m += 1

        #now we can deal with mazeMap, costMap, start, end
```

```python
    def AstarManhattanHeuristic(self):
        node_open_list = []
        node_close_list = []
        start_node = Node(self.start)
        goal_node = Node(self.end)
        node_open_list.append(start_node)

        while len(node_open_list) > 0 :
            node_open_list.sort(key=operator.attrgetter('heuristicFn'))
            current_node = node_open_list.pop(0)
            current_node.id = (current_node.value[0] * 7) + current_node.value[1];
            self.fullPath.append(current_node.id)
            node_close_list.append(current_node)
            # Check if we have reached the goal, return the path
            if current_node.value == goal_node.value:
                path = []
                self.totalCost=0
                while current_node != start_node:
                    path.append(current_node.id)
                    self.totalCost+=self.costMap[current_node.value]
                    current_node = current_node.previousNode
                path.append(0);
                return self.fullPath,path[::-1],self.totalCost

            (x, y) = current_node.value
```

```python
            # Get neighbors
            neighbors = [(x - 1, y), (x + 1, y), (x, y - 1), (x, y + 1)]
            for next in neighbors:
                if next[0]<0 or next[1]<0 or next[0]>=5 or next[1]>=7:
                    neighbors.remove(next)
            # Loop neighbors
            for next in neighbors:
                map_value = self.mazeMap.get(next)
                if (map_value == '#'):
                    continue
                neighbor = Node(next)
                neighbor.previousNode=current_node
                if (neighbor in node_close_list):
                    continue
                neighbor.gOfN=0
                neighbor.hOfN=0
                neighbor.heuristicFn=0
                neighbor.gOfN =current_node.gOfN+self.costMap[neighbor.value]
                neighbor.hOfN = abs(neighbor.value[0] - goal_node.value[0]) + abs(
                    neighbor.value[1] - goal_node.value[1])
                neighbor.heuristicFn = neighbor.gOfN + neighbor.hOfN
                if (self.not_in_open(node_open_list, neighbor) == True):
                    e =0
                    h=0
                    for node in node_open_list:
                        if neighbor.value == node.value:
                            node_open_list[h]=neighbor
                            e=1
                        h+=1
                    if(e==0):
                        node_open_list.append(neighbor)
        return None
    #end AstarManhattanHeuristic
    def not_in_open(self, open, neighbor):
        for node in open:
            if (neighbor.value == node.value and neighbor.heuristicFn >= node.heuristicFn):
                return False
        return True

# endregion
```

2.1.3. Sample run (the output)

```
**ASTAR with Manhattan Heuristic ** Full Path:[0, 7, 0, 14, 21, 7,
22, 14, 0, 29, 21, 7, 22, 14, 0, 29, 21, 7, 22, 14, 0, 29, 21, 1, 7,
22, 0, 2, 14, 29, 9, 21, 7, 16, 22, 14, 0, 17, 29, 2, 21, 9, 7, 22,
16, 18, 14, 0, 29, 17, 21, 2, 25, 7, 22, 9, 32, 31]

Path is: [0, 1, 2, 9, 16, 17, 18, 25, 32, 31]
Total Cost: 30
```

2.2. K-Nearest Neighbors

2.2.1. The main steps of the algorithm

Step 1: Handling the data
Step 2: Calculate Euclidean Distance.
Step 3: Get Nearest Neighbors.
Step 4: Make Predictions.
Step 5: Check the accuracy

2.2.2. The implementation K-NN algorithm

```python
# region KNN
class KNN_Algorithm:
    def __init__(self, K):
        self.K = K

    def euclidean_distance(self, p1, p2):
        DIST = 0.0
        for i in range(len(p1) - 1):
            DIST += (p1[i] - p2[i]) ** 2
        return sqrt(DIST)

    def fit_dataSet(self, X, y):
        self.X_train = X
        self.y_train = y

    def predict_neighbors(self, X):
        predictions_Neighbors = [self._predict(x) for x in X]
        return np.array(predictions_Neighbors)

    def _predict(self, x):
        neighborhood = [self.euclidean_distance(x, x_train) for x_train in self.X_train]
        k_indexes = np.argsort(neighborhood)[:self.K]
        output = [self.y_train[i] for i in k_indexes]
        correct_output = Counter(output).most_common(1)
        return correct_output[0][0]

    def calculate_KNN_accuracy(self, exact_y, predected_y):
        acc = (np.sum(exact_y == predected_y) / len(exact_y))*100
        return acc

    def KNN(self, X_train, X_test, Y_train, Y_test):
        self.fit_dataSet(X_train, Y_train)
        predictions_Neighbors = self.predict_neighbors(X_test)
        return self.calculate_KNN_accuracy(Y_test, predictions_Neighbors)
# endregion
```

2.2.3. Sample run (the output)

KNN Accuracy: 87. 71929824561403

2.3. Genetic Algorithm

2.3.1. The main steps of the algorithm

Step 1: Population data set
Step 2: Fitness Value
Step 3: Mating Pool
Step 4: Crossover
Step 5: Mutation

2.3.2. The implementation of Genetic algorithm

```
def fitness(self, dna):
   di=0
   for c in range(self.DNA_SIZE-1):
        di += self.cost(dna[c], dna[c+1])
   di+=self.cost(1, dna[self.DNA_SIZE-1])
   return di

def mutate(self, dna, random1, random2):
    for i in range(self.DNA_SIZE):
        if random1<0.01:
            temp = dna[int(random2)]
            dna[int(random2)]=dna[i]
            dna[i]=temp
    return dna


def crossover(self, dna1, dna2, random1, random2):
    DNA_SIZE=self.DNA_SIZE
    rr1=int(random1*DNA_SIZE)
    rr2=int(random2*DNA_SIZE)

    DNA1_1=dna1[:rr1]
    DNA1_2 =[]
    for i in range(DNA_SIZE):
        if dna2[i]not in DNA1_1:
            DNA1_2.append(dna2[i])

    DNA2_1=dna2[:rr2]
    DNA2_2 =[]
    for i in range(DNA_SIZE):
        if dna1[i]not in DNA2_1:
            DNA2_2.append(dna1[i])

    return (DNA1_1+DNA1_2,DNA2_1+DNA2_2)
```

2.3.3. Sample run (the output)

```
[1,2,3,4,5,6]
58
```

3. Discussion

3.1 A* algorithm:

Time complexity is O(E) where E is number of edges.
We can use Fibonacci heap to implement open list instead sorting the open list, then the performance will become better as Fibonacci heap takes O(1) average time to insert to open list and to decrease key.

3.2 K-NN algorithm:

Time complexity is O(n^2) where n is number of rows in the dataset.
The better K chosen, the better algorithm accuracy

3.3. Genetic algorithm:

Time complexity is O(N*L^2) where n is number of rows in the population and L is the length of the DNA, **HINT** (the complexity of fitness function is O(L^2)).
The performance of Genetic algorithm is based on :

1. The fitness function
2. The selection operator
3. The variation operators

4. References

Theory.stanford.edu. 2020. *Amit'S A* Pages*. [online] Available at: <http://theory.stanford.edu/~amitp/GameProgramming/> [Accessed 10 May 2020].

En.wikipedia.org. 2020. *A* Search Algorithm*. [online] Available at: <https://en.wikipedia.org/wiki/A*_search_algorithm> [Accessed 10 May 2020].

Yu, Xinjie, and Mitsuo Gen. Introduction to evolutionary algorithms. Springer Science & Business Media, 2010.

Gad, A., 2020. *Introduction To Optimization With Genetic Algorithm - Kdnuggets*. [online] KDnuggets. Available at: <https://www.kdnuggets.com/2018/03/introduction-optimization-with-genetic-algorithm.html> [Accessed 12 May 2020].