ربات فوتبالیستPEAS

معیار کارایی:براساس پست بازیکن‌‌؛ رسیدن به هدف طراحی شده برای مثال(حمله:گل زدن و دروازه بان:جلوگیری از گل خوردن)، تصمیم گیری درست در هر موقعیت، دقت در کنترل توپ، تعامل با هم تیمی ها، انطباق با شرایط آب و هوایی، سرعت عمل بالادر اجرا و رعایت قوانین در چارچوب یک ربات، گریز از برخورد با سطوح مخرب با دیگر بازیکنان

محیط:زمین چمن یا سالن فوتسال، محوطه های زمین، خط های اُت،آب و هوای مختلف، بازیکن های دیگر با ز، توپ، دروازه بان، داور

عملگرها:چرخ یا پای مکانیکی، سروبدن، دست، میکروفن، بلندگو، موتور

سنسورها:دوربین، ماژول وای فای، ژیروسکوپ، شاب سنج

............................................................................................................................................................................................

روشهای احتمالی برای استدلال غیرقطعی در هوش مصنوعی

بسیاری از مسائل و مشکلات در زمینه هوش مصنوعی(در استدلال، برنامه ریزی، یادگیری، درکو روباتیک) نیاز به عاملی دارد که با اطلاعات ناکافی یا غیر قطعینیز کارکند. محققان هوش مصنوعی برای حل این مشکل با بکارگیری روش هایی از نظریه احتمال و اقتصاد، ابزارهای قدرتمندی را ابداع کرده اند.

شبکه های بیزی ابزاری بسیار جامع است که می تواند برای بسیاری از مشکلات استفاده گردد:استدلال(با استفاده از الگوی استنباط بیزی)، یادگیری(با استفاده از الگوریتم امید ریاضی\_بیشینه سازی)، برنامه ریزی(با استفاده از شبکه های تصمیم گیری) و درک(با استفاده از شبکه های بیزی پویا). الگوریتم های احتمال می توانند در موارد زیر بکار گرفته شوند:فیلترینگ، پیش بینی، روان کردن و یافتن توضیحات برای جریان دادها، کمک به سیستم های درک جهت تحلیل فرایند هایی که در زمان رخ می دهند(برای مثال مدلهای مارکوف پنهان یا فیلترهای کالمن)

'مطلوبیت' مفهومی کلیدی از دانش اقتصادی است:اندازه گیری اینکه یک چیز چقدر برای عامل هوشمند، ارزشمنداست. ابزارهای دقیق ریاضی با بهره گیری از نظریه تصمیم گیری، آنالیز تصمیم گیری و نظریه ارزش اطلاعات توسعه یافته اند تا چگونگی انتخاب و برنامه ریزی عامل هوشمند را تحلیل کنند. این ابزارها شامل مدل هایی هم چون؛ فرایندهای تصمیم گیری مارکوف، شبکه های تصمیم گیری پویا، نظریه بازی و طراحی ساز و کار هستند.

................................................................................................................

مسئله 8 وزیر

import random, math

def init(pop\_size):

import itertools

permutations = itertools.permutations([1, 2, 3, 4, 5 ,6 ,7 ,8], r=None)

start = random.randint(0, 40320 - pop\_size)

stop = start + pop\_size

return list(itertools.islice(permutations, start, stop))

def print\_chrom(chrom):

for i in range(8):

print("|", end='')

for j in range(8):

if j+1 == chrom[i]:

print(" Q |", end='')

else:

print(" |", end='')

print()

print("-----------------")

def one\_queen\_penalty(index, chrom):

col = index

row = chrom[index]

penalty = 0

for i in range(len(chrom)):

if i == col:

continue

if chrom[i] < row and chrom[i] + math.fabs(col - i) == row:

penalty = penalty + 1

elif chrom[i] > row and chrom[i] - math.fabs(col - i) == row:

penalty = penalty + 1

return penalty

def configuration\_penalty(chrom):

sum = 0

for i in range(len(chrom)):

sum = sum + one\_queen\_penalty(i, chrom)

return sum

def pop\_fitness\_calculator(population):

fitness = []

for chrom in population:

penalty = configuration\_penalty(chrom)

if penalty > 0:

fitness.append(1/penalty)

else:

fitness.append(2)

return fitness

def chrom\_fitness\_calculator(chrom):

penalty = configuration\_penalty(chrom)

if penalty > 0:

return 1/penalty

else:

return 2

def selection(population, number):

randomlist = random.sample(range(0, len(population)), number)

selected = []

for i in randomlist:

selected.append(population[i])

return selected

def get\_two\_parents(population):

population.sort(reverse=True, key=chrom\_fitness\_calculator)

# print(population)

return population[0:2]

def cross\_over(parent1, parent2):

parent1 = list(parent1)

parent2 = list(parent2)

position = random.randint(1,6)

# print("Cross over Position = " , position)

child1 = parent1[0:position]

child2 = parent2[0:position]

for i in range(len(parent1)):

if parent1[i] not in child2:

child2.append(parent1[i])

if parent2[i] not in child1:

child1.append(parent2[i])

child1 = tuple(child1)

child2 = tuple(child2)

return [child1, child2]

def mutation(childs):

mutated =[]

for child in childs:

prob = random.randint(1, 100)

# print("Mutation Prob = ", prob)

if prob < mutation\_prob:

mutated.append(mutate(child))

else:

mutated.append(child)

return mutated

def mutate(chrom):

position1 = random.randint(0, len(chrom) - 1)

position2 = random.randint(0, len(chrom) - 1)

# print("Mutation Pos = ", position1, " , " , position2)

chrom = list(chrom)

temp = chrom[position1]

chrom[position1] = chrom[position2]

chrom[position2] = temp

chrom = tuple(chrom)

return chrom

def survival\_selection(population, childs):

population.sort(reverse=True, key=chrom\_fitness\_calculator)

population[-1] = childs[0]

population[-2] = childs[1]

if chrom\_fitness\_calculator(childs[0]) == 2:

return (1, population)

if chrom\_fitness\_calculator(childs[1]) == 2 :

return (2, population)

return (0, population)

pop\_size = 100

select\_random = 5

mutation\_prob = 80

pop = init(pop\_size)

rounds = 10000

TOTAL\_ROUNDS = rounds

while rounds > 0:

selected = selection(pop, select\_random)

parents = get\_two\_parents(selected)

childs = cross\_over(parents[0], parents[1])

childs = mutation(childs)

done, pop = survival\_selection(pop, childs)

if done == 1:

print("Best 10 answer after {0} iterations:".format(TOTAL\_ROUNDS - rounds))

pop.sort(reverse=True, key=chrom\_fitness\_calculator)

print(pop[0:10])

print("Best answer:")

print(childs[0])

print\_chrom(childs[0])

break

if done == 2:

print("Best 10 answer after {0} iterations:".format(TOTAL\_ROUNDS - rounds))

pop.sort(reverse=True, key=chrom\_fitness\_calculator)

print(pop[0:10])

print("Best answer:")

print(childs[1])

print\_chrom(childs[1])

break

rounds = rounds - 1

if rounds <= 0:

population.sort(reverse=True, key=chrom\_fitness\_calculator)

print("Best answer at the end : ")

print(population[0])

print\_chrom(population[0])

# selected = selection(pop, 5)

# print("##")

# print(selected)

# print("##")

# print(pop\_fitness\_calculator(selected))

# print("##")

# parents = get\_two\_parents(selected)

# print(parents)

# print("##")

# childs = cross\_over(parents[0], parents[1])

# print(childs)

# childs = mutation(childs)

# print(childs)

# print("##")

# pop.sort(reverse=True, key=chrom\_fitness\_calculator)

# print(pop[0], " = " , chrom\_fitness\_calculator(pop[0]))

# print(pop[1], " = " , chrom\_fitness\_calculator(pop[1]))

# print(pop[-1], " = " , chrom\_fitness\_calculator(pop[-1]))

# print(pop[-2], " = " , chrom\_fitness\_calculator(pop[-2]))

# print("Child 1 = " , chrom\_fitness\_calculator(childs[0]))

# print("Child 2 = " , chrom\_fitness\_calculator(childs[1]))

Best 10 answer after 168 iterations:

[(8, 4, 1, 3, 6, 2, 7, 5), (6, 1, 2, 7, 5, 3, 8, 4), (2, 6, 1, 7, 5, 3, 8, 4), (6, 3, 1, 4, 7, 8, 2, 5), (6, 3, 1, 4, 7, 8, 2, 5), (6, 8, 7, 1, 4, 2, 5, 3), (2, 5, 8, 6, 1, 3, 7, 4), (2, 5, 8, 6, 1, 3, 7, 4), (7, 3, 6, 8, 1, 4, 2, 5), (6, 1, 7, 2, 5, 3, 8, 4)]

Best answer:

(8, 4, 1, 3, 6, 2, 7, 5)

| | | | | | | | Q |

-----------------

| | | | Q | | | | |

-----------------

| Q | | | | | | | |

-----------------

| | | Q | | | | | |

-----------------

| | | | | | Q | | |

-----------------

| | Q | | | | | | |

-----------------

| | | | | | | Q | |

-----------------

| | | | | Q | | | |

-----------------

print\_chrom((6, 3, 1, 2, 5, 8, 4, 7))

| | | | | | Q | | |

-----------------

| | | Q | | | | | |

-----------------

| Q | | | | | | | |

-----------------

| | Q | | | | | | |

-----------------

| | | | | Q | | | |

-----------------

| | | | | | | | Q |

-----------------

| | | | Q | | | | |

-----------------

| | | | | | | Q | |

-----------------

........................................................................................................................................................................................