

**ITCS 6150-INTELLIGENT SYSTEMS**

**SOLVING 8 QUENNS USING SIMULATED ANNEALING**

**Submitted By Instructor**

Sanket Revadigar(801203510) Prof. Dewan T. Ahmed, Ph.D.

**I.SIMULATED ANNEALING**

Simulated annealing is a probabilistic global search optimization algorithm. This means that randomness is used as part of the search process. This makes this algorithm suitable for nonlinear objective functions where other local search algorithms do not work well.

**II.PSEUDOCODE**

Let *s* = *s*0

For *k* = 0 through *k*max (exclusive):

*T* ← temperature( 1 - *(k+1)*/*k*max )

Pick a random neighbour, *s*new ← neighbour(*s*)

If *P*(*E*(*s*), *E*(*s*new), *T*) ≥ random(0, 1):

*s* ← *s*new

Output: the final state *s*

**III.IMPLEMENTATION**

1. Randomly move or change the state
2. Evaluate the energy of the new state using the objective function
3. Compares energy to previous conditions and decides whether to accept or reject the new solution based on the current temperature
4. Repeat this until you have converged on an acceptable answer
5. One of two requirements must be met for the move to be accepted
6. Movement causes a decrease in state energy (ie, an improvement in the objective function).
7. The move increases state energy (that is, a slightly worse solution), but within the temperature range. As the algorithm progresses, the temperature drops exponentially. In this way, avoids getting caught in the local minimums early in the process and instead start working on a feasible solution at the end.

**IV.PARAMETERS**

There are total 2 classes(Initial state,simulated\_annealing)

**Initial state**-Initializes the 8 queen problem randomly

reset\_board-Resets the 8 queen board

cost\_value-Calculates the cost to make a move

cost\_value\_queen-Costs of each individual queen is calculated

to\_string-returns the board state in string format

lowest\_value\_state-returns the state with the lowest cost

**simulated\_annealing**-Solves the 8 queens problem using annealing

\_\_init\_\_-Initializes the variables

start-starts the simulated annealing

time\_spent-calculates the total time spent for the code to execute

**V.CODE**

import random, math

import decimal

#from copy import deepcopy, copy

from datetime import datetime

#initalise the 8 queens problem randomly

class Initial\_state:

def \_\_init\_\_(self, total\_queens=8):

self.total\_queens = total\_queens

self.reset\_board()

#fucntion to reset the board

def reset\_board(self):

self.queens = [-1 for i in range(0, self.total\_queens)]

for i in range(0, self.total\_queens):

self.queens[i] = random.randint(0, self.total\_queens - 1)

# self.queens[row] = column

#calculates the cost

def cost\_value(self):

attack = 0

for queen in range(0, self.total\_queens):

for queen\_next in range(queen+1, self.total\_queens):

if self.queens[queen] == self.queens[queen\_next] or abs(queen - queen\_next) == abs(self.queens[queen] - self.queens[queen\_next]):

attack += 1

return attack

@staticmethod

def cost\_value\_queen(queens):

attack = 0

total\_queens = len(queens)

for queen in range(0, total\_queens):

for queen\_next in range(queen+1, total\_queens):

if queens[queen] == queens[queen\_next] or abs(queen - queen\_next) == abs(queens[queen] - queens[queen\_next]):

attack += 1

return attack

@staticmethod

def to\_string(queens):

board\_To\_string = ""

for row, column in enumerate(queens):

board\_To\_string += "(%s, %s)\n\n" % (row, column)

return board\_To\_string

#calculate the lowest value of the state

def lowest\_value\_state(self):

#disp\_count = 0

temporary\_queens = self.queens

#lowest\_cost = self.cost\_value(temporary\_queens)

for i in range(0, self.total\_queens):

temporary\_queens[i] = (temporary\_queens[i] + 1) % (self.total\_queens - 1)

for j in range(self.total\_queens):

temporary\_queens[j] = (temporary\_queens[j] + 1) % (self.total\_queens - 1)

def \_\_str\_\_(self):

board\_To\_string = ""

for row, column in enumerate(self.queens):

board\_To\_string += "(%s, %s)\n\n" % (row, column)

return board\_To\_string

class simulated\_annealing:

def \_\_init\_\_(self, state):

self.timespent = 0;

self.state = state

self.temperature = 4000

self.sch = 0.99

self.start\_time = datetime.now()

def start(self):

state = self.state

board\_queens = self.state.queens[:]

solution = False

for k in range(0, 170000):

self.temperature \*= self.sch

state.reset\_board()

successor\_queens = state.queens[:]

dw = Initial\_state.cost\_value\_queen(successor\_queens) - Initial\_state.cost\_value\_queen(board\_queens)

exp = decimal.Decimal(decimal.Decimal(math.e) \*\* (decimal.Decimal(-dw) \* decimal.Decimal(self.temperature)))

if dw > 0 or random.uniform(0, 1) < exp:

board\_queens = successor\_queens[:]

if Initial\_state.cost\_value\_queen(board\_queens) == 0:

print("Goal\_state:")

print(Initial\_state.to\_string(board\_queens))

self.timespent = self.time\_spent()

print("Successful\nTime Spent : %sms" % (str(self.timespent)))

solution = True

break

if solution == False:

self.timespent = self.time\_spent()

print("Unsuccessful\nTime Spent : %sms" % (str(self.timespent)))

return self.timespent

def time\_spent(self):

end\_time = datetime.now()

timespent = (end\_time - self.start\_time).microseconds / 1000

return timespent

if \_\_name\_\_ == '\_\_main\_\_':

state = Initial\_state()

print("Initial\_state:")

print(state)

simulated\_annealing(state).start()

**VI.SAMPLE OUTPUTS**

**Output 1:**

Initial\_state:

(0, 0)

(1, 4)

(2, 5)

(3, 1)

(4, 0)

(5, 4)

(6, 7)

(7, 1)

Goal\_state:

(0, 4)

(1, 1)

(2, 5)

(3, 0)

(4, 6)

(5, 3)

(6, 7)

(7, 2)

Successful

Time Spent : 259.773ms

**Output 2:**

Initial\_state:

(0, 7)

(1, 5)

(2, 2)

(3, 7)

(4, 4)

(5, 6)

(6, 0)

(7, 7)

Unsuccessful

Time Spent : 950.298ms