**Questions:**

1. **With the help of the supplementary materials and demonstrated codes implement the variants of hill-climbing and genetic algorithms discussed above in Prolog and Python.**

**Solution to the question no. 1(a):**

**The demonstrated Prolog code of the Random restart hill climbing Algorithm is as below:**

:-use\_module(eval\_state).

:-use\_module(list\_apps).

:-dynamic(state/4). /\* id,type,state,h\_value\*/

:-dynamic(id/1).

:-dynamic(max\_val/1).

:-dynamic(threshold/1).

:-dynamic(restrt\_cntr/1).

:-dynamic(list\_st/1).

/\* Organizing a Menu \*/

start:- repeat,

write('\n1. Clear database'),

write('\n2. Execute hcls'),

write('\n3. Display states'),

write('\n4. Save states'),

write('\n5. Exit'),

write('\n\nEnter your choice: '),

read(N), N >0, N < 6,

do(N), N=5,!.

do(1):- retractall(state(\_,\_,\_,\_)),retractall(id(\_)), retractall(max\_val(\_)),

retractall(threshold(\_)), retractall(restrt\_cntr(\_)).

do(2):- go\_hcs.

do(3):- listing(state).

do(4):- write('Enter a new file name:'), read(Flnm),

tell(Flnm),listing(state),told.

do(5):- abort.

/\* Beginning of search \*/

go\_hcs:- write('Enter a state:'), read(S),

write('Enter threshold value:'), read(V),

assert(threshold(V)),assert(restrt\_cntr(0)),

getdigits(S,D1,D2,D3,D4,D5,D6,D7,D8),

L=[D1,D2,D3,D4,D5,D6,D7,D8],

gnrt\_sucsr(L).

/\* Generating the successors of a 8-queens' state given as a list \*/

gnrt\_sucsr(L):- assert(id(1)), assert(state(1,'c',L,50)),

incr\_id, mk\_new(1,L), retract(id(\_)), evaluate.

incr\_id:-id(V), V1 is V+1, retract(id(\_)), assert(id(V1)).

mk\_new(9,\_):-!.

mk\_new(N,L):- nthel(N,L,X), del\_el(X,[1,2,3,4,5,6,7,8],L1),

cng\_mk(N,L,L1), N1 is N+1, mk\_new(N1,L).

cng\_mk(\_,\_,[]):-!.

cng\_mk(N,L,L1):- L1=[H|T], rplc\_nthel(N,H,L,L2),id(Id),

assert(state(Id,'s',L2,50)), incr\_id, cng\_mk(N,L,T).

/\* Evaluating the states \*/

evaluate:- eval\_all, checkall.

eval\_all:- state(I,T,L,\_), eval(L,V),retract(state(I,\_,\_,\_)),

assert(state(I,T,L,V)), fail.

eval\_all:-!.

/\* Determining and displaying the best state \*/

checkall:- state(\_,'c',\_,V1), threshold(V2), V1 >= V2, I is 1, dsply(I),!.

checkall:- best(I1,V1), threshold(V2), V1 >= V2, I is I1, dsply(I),!.

checkall:- state(\_,'c',\_,V1), best(I,V2) ,V2>V1,state(I,\_,L,\_),

retractall(state(\_,\_,\_,\_)),write\_list(['\nIteration max: ',V2]),

gnrt\_sucsr(L),!.

checkall:- rndm\_restrt,!.

best(I,Max):- state(\_,'s',\_,Val), assert(max\_val(Val)),

updt\_max, max\_val(Max), state(I,\_,\_,Max), retract(max\_val(\_)),!.

updt\_max:- state(\_,\_,\_,V2), max\_val(V1), V2>V1,

retract(max\_val(\_)), assert(max\_val(V2)), fail.

updt\_max:-!.

dsply(I):-state(I,T,L,V),

write\_list(['\n\nFound! Id:',I,' ',T,' ', L,' ','Value:',V,'\n']),!.

rndm\_restrt:- retractall(state(\_,\_,\_,\_)), restrt\_cntr(V), V<5,

write('\n\nStuckup! Restarting.\n\n'), restart,!.

rndm\_restrt:-write('\n\nStuckup once again! Ending.\n\n').

restart:- incr\_r\_c, restrt\_cntr(V), write\_list(['\n\nRestart index: ',V]),

get\_rndm\_st(L),gnrt\_sucsr(L).

incr\_r\_c:- restrt\_cntr(V), V1 is V+1, retract(restrt\_cntr(\_)),

assert(restrt\_cntr(V1)).

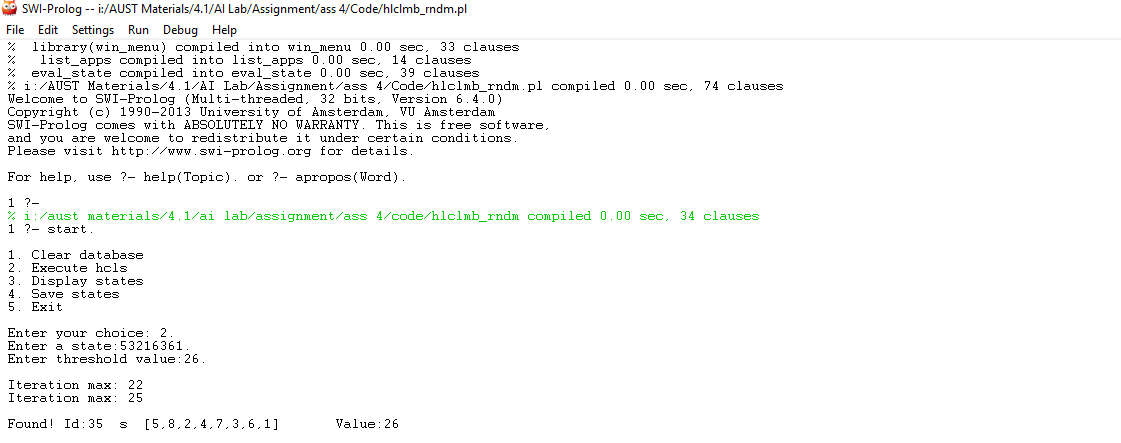
get\_rndm\_st(L):- assert(list\_st([])), lp8(8), list\_st(L), retract(list\_st(\_)).

lp8(0):-!.

lp8(N):- N1 is N-1, X is random(8)+1, list\_st(L1),append(L1,[X],L2),

retract(list\_st(L1)),assert(list\_st(L2)),lp8(N1).

**A sample input and output is as below:**



**The demonstrated Python code of the Random restart hill climbing Algorithm is as below:**

import random

threshold = 0

max\_val = 0

restart\_count = 0

states = []

max\_id = 0

def clear\_database():

global threshold

global max\_val

global restart\_count

global states

global max\_id

threshold = 0

max\_val = 0

restart\_count = 0

states = []

max\_id = 0

def execute\_hcls():

global threshold

global states

state = input('enter a state: ')

threshold = int(input('enter threshold value: '))

L = list(state)

genarate\_successor(L)

def display\_states():

global states

for j in range(len(states)):

print(states[j])

def save\_states():

global states

f1=open("std.py", "w")

for j in range(len(states)):

print(states[j],file = f1)

f1.close

def evalState(qList):

count = 0

#horizontal check

for i in range(0,len(qList),1):

for j in range(i+1,len(qList),1):

if (qList[i] == qList[j]):

count = count + 1

#diagonal\_up check

for i in range(0,len(qList),1):

x = int(qList[i])

for j in range(i+1,len(qList),1):

x = x + 1

if (x == int(qList[j])):

count = count + 1

#diagonal\_down check

for i in range(0,len(qList),1):

x = int(qList[i])

for j in range(i+1,len(qList),1):

x = x - 1

if (x == int(qList[j])):

count = count + 1

return count

def eval(L):

return 28 - evalState(L)

def genarate\_successor(L):

global threshold

global max\_val

global restart\_count

global states

global max\_id

hval = eval(L)

y = [1,'c',L,hval]

#print(y)

stuck = True

max\_val = hval

max\_id = 1

states = []

states.append(y)

idcount = 2

for i in range(1,9):

for j in range(1,9):

if(L[i-1] != str(j)):

x = L[:]

x[i-1] = str(j)

hval = eval(x)

y = [idcount,'s',x,hval]

states.append(y)

if(max\_val < hval):

max\_val = hval

max\_id = idcount

#print(max\_val,end =' ')

#print(max\_id)

stuck = False

idcount += 1

checkall(stuck)

def checkall(stuck):

global threshold

global max\_val

global restart\_count

global states

global max\_id

if max\_val >= threshold:

print('found! '+str(states[max\_id - 1]))

elif(restart\_count >= 5):

print('Stuckup Again! Stop..\n ')

elif stuck == True and restart\_count < 5:

print('Stuckup! Restarting..\n ')

restart\_count += 1

genarate\_successor(randomState())

elif max\_val < threshold :

print('iteration max: '+str(max\_val))

genarate\_successor(states[max\_id - 1][2])

def randomState():

rstate = []

for x in range(8):

val = str(random.randint(1,9))

rstate.append(val)

return rstate

#Main

cs = 1

while(cs>=1 and cs<5):

print('\n1. Clear database')

print('\n2. Execute hcls')

print('\n3. Display states')

print('\n4. Save states')

print('\n5. Exit')

cs = int(input('\n\nEnter your choice: '))

if(cs == 1):

clear\_database()

elif(cs == 2):

execute\_hcls()

elif(cs == 3):

display\_states()

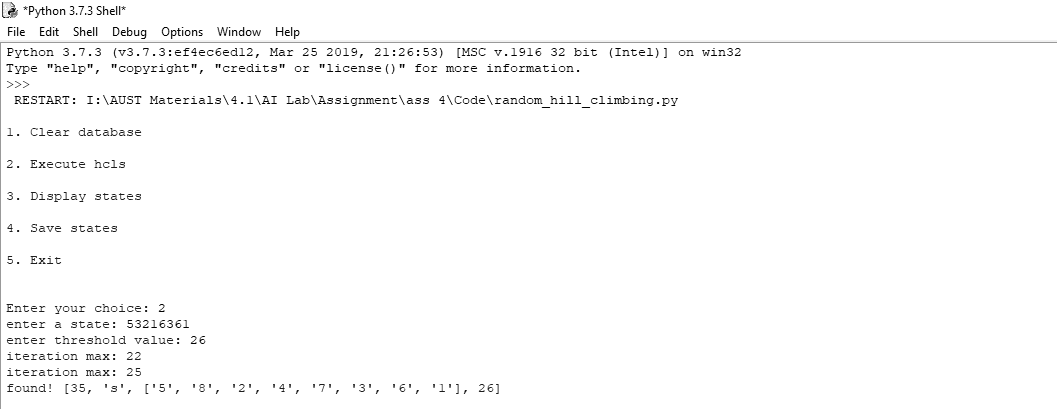
elif(cs == 4):

save\_states()

else:

break

**A sample input and output is as below:**



**Solution to the question no. 1(b):**

**The demonstrated Prolog code of the Stochastic Hill-climbing Algorithm is as below:**

:-use\_module(eval\_state).

:-use\_module(list\_apps).

:-dynamic(state/4). /\* id,type,state,h\_value\*/

:-dynamic(id/1).

:-dynamic(max\_val/1).

:-dynamic(threshold/1).

:-dynamic(restrt\_cntr/1).

:-dynamic(list\_st/1).

/\* Organizing a Menu \*/

start:- repeat,

write('\n1. Clear database'),

write('\n2. Execute hcls'),

write('\n3. Display states'),

write('\n4. Save states'),

write('\n5. Exit'),

write('\n\nEnter your choice: '),

read(N), N >0, N < 6,

do(N), N=5,!.

do(1):- retractall(state(\_,\_,\_,\_)),retractall(id(\_)), retractall(max\_val(\_)),

retractall(threshold(\_)), retractall(restrt\_cntr(\_)).

do(2):- go\_hcs.

do(3):- listing(state).

do(4):- write('Enter a new file name:'), read(Flnm),

tell(Flnm),listing(state),told.

do(5):- abort.

/\* Beginning of search \*/

go\_hcs:- write('Enter a state:'), read(S),

write('Enter threshold value:'), read(V),

assert(threshold(V)),assert(restrt\_cntr(0)),

getdigits(S,D1,D2,D3,D4,D5,D6,D7,D8),

L=[D1,D2,D3,D4,D5,D6,D7,D8],

gnrt\_sucsr(L).

/\* Generating the successors of a 8-queens' state given as a list \*/

gnrt\_sucsr(L):- assert(id(1)), assert(state(1,'c',L,50)),

incr\_id, mk\_new(1,L), retract(id(\_)), evaluate.

incr\_id:-id(V), V1 is V+1, retract(id(\_)), assert(id(V1)).

mk\_new(9,\_):-!.

mk\_new(N,L):- nthel(N,L,X), del\_el(X,[1,2,3,4,5,6,7,8],L1),

cng\_mk(N,L,L1), N1 is N+1, mk\_new(N1,L).

cng\_mk(\_,\_,[]):-!.

cng\_mk(N,L,L1):- L1=[H|T], rplc\_nthel(N,H,L,L2),id(Id),

assert(state(Id,'s',L2,50)), incr\_id, cng\_mk(N,L,T).

/\* Evaluating the states \*/

evaluate:- eval\_all,plist.

eval\_all:- state(I,T,L,\_), eval(L,V),retract(state(I,\_,\_,\_)),

assert(state(I,T,L,V)), fail.

eval\_all:-!.

dsply(I):-state(I,T,L,V),

write\_list(['\n\nFound! Id:',I,' ',T,' ', L,' ','Value:',V,'\n']),!.

plist:- state(\_,'c',\_,V1), threshold(V2), V1 >= V2, I is 1, dsply(I),!.

plist:- best(I1,V1), threshold(V2), V1 >= V2, I is I1, dsply(I),!.

plist:-state(\_,c,\_,V),state(I1,s,L,V1),V1>V,retractall(state(\_,\_,\_,\_)),

X is random(2)+1, X>1,assert(list\_st([L])), write('Random Uphill: ') ,write\_list([I1,' ',L,' ',V1]),nl, gnrt\_sucsr(L), !.

plist:- rndm\_restrt,!.

best(I,Max):- state(\_,'s',\_,Val), assert(max\_val(Val)),

updt\_max, max\_val(Max), state(I,\_,\_,Max), retract(max\_val(\_)),!.

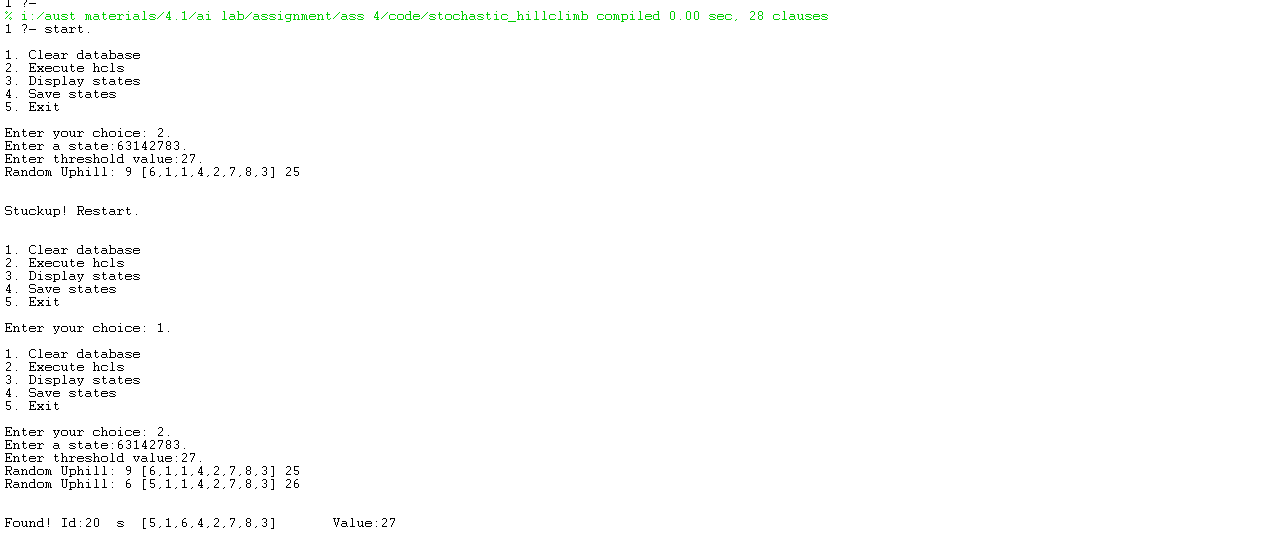
updt\_max:- state(\_,\_,\_,V2), max\_val(V1), V2>V1,

retract(max\_val(\_)), assert(max\_val(V2)), fail.

updt\_max:-!.

rndm\_restrt:- retractall(state(\_,\_,\_,\_)), write('\n\nStuckup! Restart.\n\n').

**A sample input and output is as below:**



**The demonstrated Python code of the Stochastic Hill-climbing Algorithm is as below:**

import random

threshold = 0

max\_val = 0

restart\_count = 0

states = []

max\_id = 0

def clear\_database():

global threshold

global max\_val

global restart\_count

global states

global max\_id

threshold = 0

max\_val = 0

restart\_count = 0

states = []

max\_id = 0

def execute\_hcls():

global threshold

global states

state = input('enter a state: ')

threshold = int(input('enter threshold value: '))

L = list(state)

genarate\_successor(L)

def display\_states():

global states

for j in range(len(states)):

print(states[j])

def save\_states():

global states

f1=open("std.py", "w")

for j in range(len(states)):

print(states[j],file = f1)

f1.close

def evalState(qList):

count = 0

#horizontal check

for i in range(0,len(qList),1):

for j in range(i+1,len(qList),1):

if (qList[i] == qList[j]):

count = count + 1

#diagonal\_up check

for i in range(0,len(qList),1):

x = int(qList[i])

for j in range(i+1,len(qList),1):

x = x + 1

if (x == int(qList[j])):

count = count + 1

#diagonal\_down check

for i in range(0,len(qList),1):

x = int(qList[i])

for j in range(i+1,len(qList),1):

x = x - 1

if (x == int(qList[j])):

count = count + 1

return count

def eval(L):

return 28 - evalState(L)

def genarate\_successor(L):

global threshold

global max\_val

global restart\_count

global states

global max\_id

isFound = False

hval = eval(L)

y = [1,'c',L,hval]

#print(y)

stuck = True

max\_val = hval

max\_id = 1

states = []

states.append(y)

if max\_val >= threshold:

isFound = True

checkall(stuck)

if isFound == False:

idcount = 2

uphills = []

for i in range(1,9):

for j in range(1,9):

if(L[i-1] != str(j)):

x = L[:]

x[i-1] = str(j)

hval = eval(x)

y = [idcount,'s',x,hval]

states.append(y)

if(max\_val < hval):

uphills.append(idcount)

stuck = False

idcount += 1

if(stuck == False and len(uphills)>0):

uid = random.randint(1,len(uphills)) - 1

max\_id = uphills[ uid ]

max\_val = states[uid - 1][3]

checkall(stuck)

def checkall(stuck):

global threshold

global max\_val

global restart\_count

global states

global max\_id

if max\_val >= threshold:

print('found! '+str(states[max\_id - 1]))

elif stuck == True:

print('Stuckup!\n ')

elif max\_val < threshold :

print('Selected Uphill: '+str(states[max\_id - 1]))

genarate\_successor(states[max\_id - 1][2])

#Main

cs = 1

while(cs>=1 and cs<5):

print('\n1. Clear database')

print('\n2. Execute hcls')

print('\n3. Display states')

print('\n4. Save states')

print('\n5. Exit')

cs = int(input('\n\nEnter your choice: '))

if(cs == 1):

clear\_database()

elif(cs == 2):

execute\_hcls()

elif(cs == 3):

display\_states()

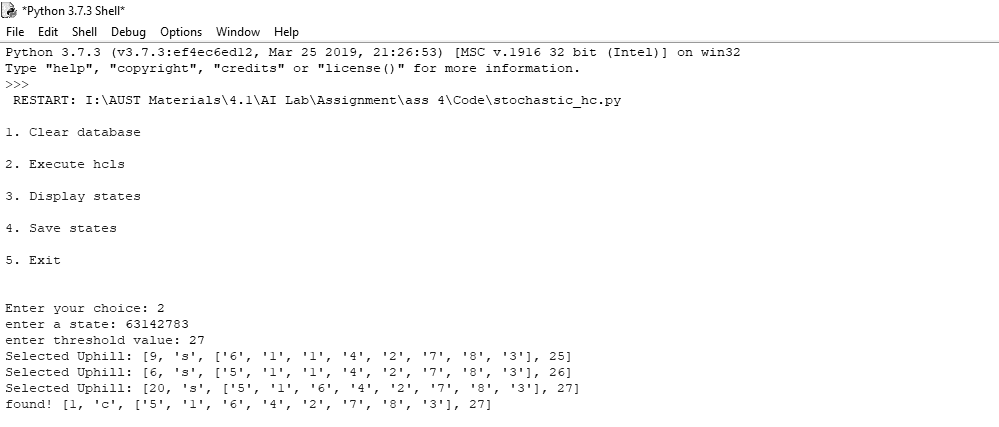
elif(cs == 4):

save\_states()

else:

break

**A sample input and output is as below:**



**Solution to the question no. 1(c):**

**The demonstrated Prolog code of the Simulated annealing Algorithm is as below:**

:-use\_module(eval\_state).

:-use\_module(list\_apps).

:-dynamic(state/4). /\* id,type,state,h\_value\*/

:-dynamic(id/1).

:-dynamic(max\_val/1).

:-dynamic(threshold/1).

:-dynamic(restrt\_cntr/1).

:-dynamic(list\_st/1).

/\* Organizing a Menu \*/

start:- repeat,

write('\n1. Clear database'),

write('\n2. Execute hcls'),

write('\n3. Display states'),

write('\n4. Save states'),

write('\n5. Exit'),

write('\n\nEnter your choice: '),

read(N), N >0, N < 6,

do(N), N=5,!.

do(1):- retractall(state(\_,\_,\_,\_)),retractall(id(\_)), retractall(max\_val(\_)),

retractall(threshold(\_)), retractall(restrt\_cntr(\_)).

do(2):- go\_hcs.

do(3):- listing(state).

do(4):- write('Enter a new file name:'), read(Flnm),

tell(Flnm),listing(state),told.

do(5):- abort.

/\* Beginning of search \*/

go\_hcs:- write('Enter a state:'), read(S),

write('Enter threshold value:'), read(V),

assert(threshold(V)),assert(restrt\_cntr(0)),

getdigits(S,D1,D2,D3,D4,D5,D6,D7,D8),

L=[D1,D2,D3,D4,D5,D6,D7,D8],

gnrt\_sucsr(L).

/\* Generating the successors of a 8-queens' state given as a list \*/

gnrt\_sucsr(L):- assert(id(1)), assert(state(1,'c',L,50)),

incr\_id, mk\_new(1,L), retract(id(\_)), evaluate.

incr\_id:-id(V), V1 is V+1, retract(id(\_)), assert(id(V1)).

mk\_new(9,\_):-!.

mk\_new(N,L):- nthel(N,L,X), del\_el(X,[1,2,3,4,5,6,7,8],L1),

cng\_mk(N,L,L1), N1 is N+1, mk\_new(N1,L).

cng\_mk(\_,\_,[]):-!.

cng\_mk(N,L,L1):- L1=[H|T], rplc\_nthel(N,H,L,L2),id(Id),

assert(state(Id,'s',L2,50)), incr\_id, cng\_mk(N,L,T).

/\* Evaluating the states \*/

evaluate:- eval\_all,checkall.

eval\_all:- state(I,T,L,\_), eval(L,V),retract(state(I,\_,\_,\_)),

assert(state(I,T,L,V)), fail.

eval\_all:-!.

dsply(I):-state(I,T,L,V),

write\_list(['\n\nFound! Id:',I,' ',T,' ', L,' ','Value:',V,'\n']),!.

checkall:- state(\_,'c',\_,V1), threshold(V2), V1 >= V2, I is 1, dsply(I),!.

checkall:- best(I1,V1), threshold(V2), V1 >= V2, I is I1, dsply(I),!.

checkall:- incr\_c, divideBy(X1),not(X1=0), state(\_,c,\_,V),state(I1,s,L,V1),V1>V, retractall(state(\_,\_,\_,\_)),

X is random(2)+1, X>1 ,assert(list\_st([L])), write('Random Uphill Successor : ') ,write\_list([I1,' ',L,' ',V1]),nl, gnrt\_sucsr(L), !.

checkall:- divideBy(X1), X1=0 , state(\_,c,\_,V),state(I1,s,L,V1), V1<V, retractall(state(\_,\_,\_,\_)),

X is random(2)+1, X>1 ,assert(list\_st([L])), write('Random Downhill Successor : ') ,write\_list([I1,' ',L,' ',V1]),nl, gnrt\_sucsr(L), !.

checkall:- rndm\_restrt,!.

best(I,Max):- state(\_,'s',\_,Val), assert(max\_val(Val)),

updt\_max, max\_val(Max), state(I,\_,\_,Max), retract(max\_val(\_)),!.

updt\_max:- state(\_,\_,\_,V2), max\_val(V1), V2>V1,

retract(max\_val(\_)), assert(max\_val(V2)), fail.

updt\_max:-!.

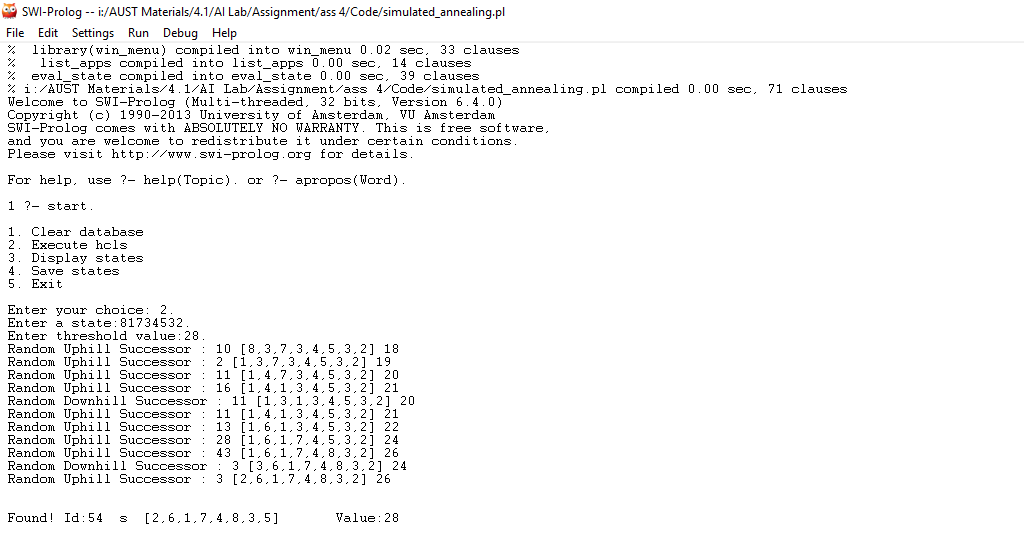
rndm\_restrt:- retractall(state(\_,\_,\_,\_)), write('\n\nStuckup! Restart.\n\n').

incr\_c:- restrt\_cntr(V), V1 is V+1, retract(restrt\_cntr(\_)),

assert(restrt\_cntr(V1)).

divideBy(X1) :- restrt\_cntr(V), X1 is V mod 5,!.

**A sample input and output is as below:**



**The demonstrated Python code of the Simulated annealing Algorithm is as below:**

import random

threshold = 0

max\_val = 0

counter = 1

states = []

max\_id = 0

def clear\_database():

global threshold

global max\_val

global counter

global states

global max\_id

threshold = 0

max\_val = 0

counter = 1

states = []

max\_id = 0

def execute\_hcls():

global threshold

global states

state = input('enter a state: ')

threshold = int(input('enter threshold value: '))

L = list(state)

genarate\_successor(L)

def display\_states():

global states

for j in range(len(states)):

print(states[j])

def save\_states():

global states

f1=open("std.py", "w")

for j in range(len(states)):

print(states[j],file = f1)

f1.close

def evalState(qList):

count = 0

#horizontal check

for i in range(0,len(qList),1):

for j in range(i+1,len(qList),1):

if (qList[i] == qList[j]):

count = count + 1

#diagonal\_up check

for i in range(0,len(qList),1):

x = int(qList[i])

for j in range(i+1,len(qList),1):

x = x + 1

if (x == int(qList[j])):

count = count + 1

#diagonal\_down check

for i in range(0,len(qList),1):

x = int(qList[i])

for j in range(i+1,len(qList),1):

x = x - 1

if (x == int(qList[j])):

count = count + 1

return count

def eval(L):

return 28 - evalState(L)

def genarate\_successor(L):

global threshold

global max\_val

global counter

global states

global max\_id

isFound = False

hval = eval(L)

y = [1,'c',L,hval]

#print(y)

stuck = True

max\_val = hval

max\_id = 1

states = []

states.append(y)

if max\_val >= threshold:

isFound = True

checkall(stuck)

if isFound == False:

idcount = 2

uphills = []

for i in range(1,9):

for j in range(1,9):

if(L[i-1] != str(j)):

x = L[:]

x[i-1] = str(j)

hval = eval(x)

y = [idcount,'s',x,hval]

states.append(y)

if(divideBy(counter) != 0 and max\_val < hval):

uphills.append(idcount)

stuck = False

elif(divideBy(counter) == 0 and max\_val > hval):

uphills.append(idcount)

stuck = False

idcount += 1

if(stuck == False and len(uphills)>0):

uid = random.randint(1,len(uphills)) - 1

max\_id = uphills[ uid ]

max\_val = states[uid - 1][3]

checkall(stuck)

def checkall(stuck):

global threshold

global max\_val

global counter

global states

global max\_id

#print('counter : '+str(counter))

if max\_val >= threshold:

print('found! '+str(states[max\_id - 1]))

elif stuck == True:

print('Stuckup!\n ')

elif max\_val < threshold and stuck == False :

if divideBy(counter) != 0:

print('Selected Uphill: '+str(states[max\_id - 1]))

counter += 1

genarate\_successor(states[max\_id - 1][2])

else:

print('Selected Downhill: '+str(states[max\_id - 1]))

counter += 1

genarate\_successor(states[max\_id - 1][2])

def divideBy(x):

return x % 5

#Main

cs = 1

while(cs>=1 and cs<5):

print('\n1. Clear database')

print('\n2. Execute hcls')

print('\n3. Display states')

print('\n4. Save states')

print('\n5. Exit')

cs = int(input('\n\nEnter your choice: '))

if(cs == 1):

clear\_database()

elif(cs == 2):

execute\_hcls()

elif(cs == 3):

display\_states()

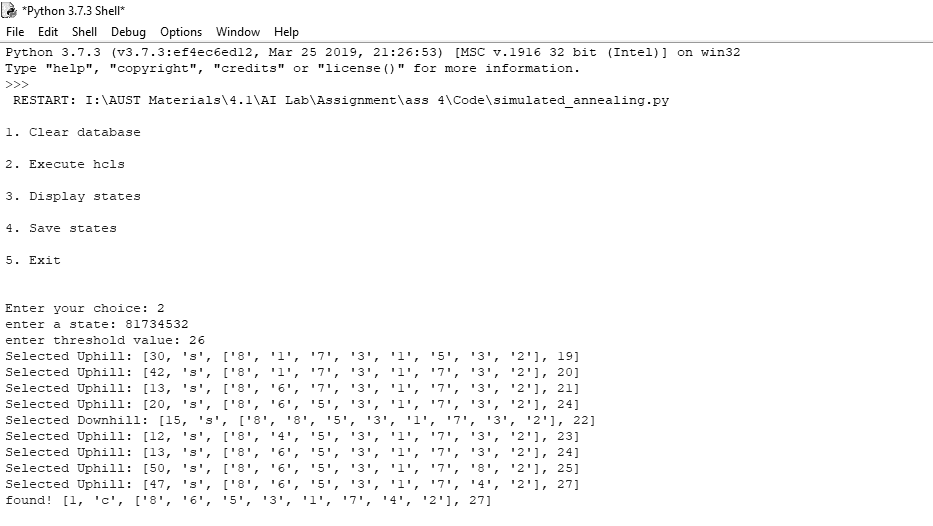
elif(cs == 4):

save\_states()

else:

break

**A sample input and output is as below:**



**Solution to the question no. 1(d):**

**The demonstrated Python code of the Genetic Algorithm is as below:**

import random

threshold = 0

max\_val = 0

restart\_count = 0

states = []

max\_id = 0

def clear\_database():

global threshold

global max\_val

global restart\_count

global states

global max\_id

threshold = 0

max\_val = 0

restart\_count = 0

states = []

max\_id = 0

def execute\_hcls():

global threshold

global states

n = input('How many parent generation: ')

for i in range(int(n)):

state = input('enter state '+str(i+1)+' : ')

L = list(state)

states.append(L)

threshold = int(input('enter threshold value: '))

makeParents()

def display\_states():

global states

for j in range(len(states)):

print(states[j])

def save\_states():

global states

f1=open("std.py", "w")

for j in range(len(states)):

print(states[j],file = f1)

f1.close

def evalState(qList):

count = 0

#horizontal check

for i in range(0,len(qList),1):

for j in range(i+1,len(qList),1):

if (qList[i] == qList[j]):

count = count + 1

#diagonal\_up check

for i in range(0,len(qList),1):

x = int(qList[i])

for j in range(i+1,len(qList),1):

x = x + 1

if (x == int(qList[j])):

count = count + 1

#diagonal\_down check

for i in range(0,len(qList),1):

x = int(qList[i])

for j in range(i+1,len(qList),1):

x = x - 1

if (x == int(qList[j])):

count = count + 1

return count

def eval(L):

return 28 - evalState(L)

def makeParents():

global states

global threshold

maxValue = 0

maxState = []

isFound = False

stuck = False

i = 0

while i < len(states):

if eval(states[i]) >= threshold:

maxState = states[i]

maxValue = eval(states[i])

isFound = True

break

elif eval(states[i]) < 15:

states.remove(states[i])

else:

maxValue = max(maxValue, eval(states[i]))

i += 1

if len(states) == 0:

stuck = True

#Chossing parents randomly for crossover

if isFound == False and stuck == False:

"""

print('Parent Generations: \n')

for i in range( len(states) ):

print(states[i])

"""

x = random.randint(0,len(states)-1)

while True:

y = random.randint(0,len(states)-1)

if(y != x):

break

parent1 = states[x]

parent2 = states[y]

crossover(parent1, parent2)

if isFound == True:

print('Found! '+str(maxState)+' '+str(maxValue))

elif stuck == True:

print('Stuck! ')

else:

print('Iteration Max: '+str(maxValue))

makeParents()

def crossover(parent1, parent2):

global states

crossoverPoint = random.randint(0,8)

offSpring1 = []

offSpring2 = []

offSpring1[: crossoverPoint] = parent1[ :crossoverPoint ]

offSpring1[crossoverPoint: ] = parent2[crossoverPoint : ]

offSpring2[: crossoverPoint] = parent2[ :crossoverPoint ]

offSpring2[crossoverPoint: ] = parent1[crossoverPoint : ]

#Mutation

if random.randint(0,10) > 7:

x = random.randint(0,7)

offSpring1[x] = str(random.randint(1,8))

states.append(offSpring1)

states.append(offSpring2)

#Main

cs = 1

while(cs>=1 and cs<5):

print('\n1. Clear database')

print('\n2. Execute hcls')

print('\n3. Display states')

print('\n4. Save states')

print('\n5. Exit')

cs = int(input('\n\nEnter your choice: '))

if(cs == 1):

clear\_database()

elif(cs == 2):

execute\_hcls()

elif(cs == 3):

display\_states()

elif(cs == 4):

save\_states()

else:

break

**A sample input and output is as below:**

