CS571 AI LAB 05

02nd October 2022

Ishita Singh 1901CS27
Kavya Goyal 1901CS30
Priyanka Sachan 1901CS43

Team Code: 1901cs30_1901cs27_1901cs43
Colab Notebook Link

OBJECTIVE

The objective of this assignment was to implement Genetic Algorithm for the 8-puzzle problem. The following things were to be taken into consideration.

1) Initial Population (assume to be 10)
The initial population of size 10 is found by extrapolating the start state

```
_population = [individual(init_board, list()) for _ in range(INIT_POPULATION)]
```

2) Selection (use Roulette Wheel Selection)
The selection of the next generation of parents is done via selection of parents based on their fitness function and modeling in the roulette wheel

```
def spin_the_wheel(wheel_size, cost_invs):
  spin_val = random.uniform(0.0,1) * wheel_size
  for i, cost_inv_lim in enumerate(cost_invs):
    if spin_val < cost_inv_lim:
      return i
```

3) Crossover (high probability value to be chosen, usually above 0.6)
The crossover is between the parents. The mating function splits the genes

```
def cross_over():
  num_cross_over = math.ceil(len(population)*CROSSOVER)
  parents = list(range(len(population)))
  for i in range(int(num_cross_over)):
      a, b = random.sample(parents, 2)
      if a != b:
        parents.remove(a)
        parents.remove(b)
      if len(population[a].genes)>0 and len(population[b].genes)>0 :
        A, B = mate(population[a], population[b])
        population.append(A)
        population.append(B)
```

4) Mutation (low probability value to be chosen, usually below 0.2)

```
def perform_mutations():
  for p in population:
    if random.random() < MUTATION:
      mutate(p)
```

5) Fitness function: Devise your own two fitness functions.
For the purpose of this assignment, two fitness functions are used

- h1 Heuristic: number of misplaced tiles
- h2 Heuristic: manhattan distances

The task is deemed complete when we get a child in the population such that its fitness function value is 0 and hence is the target state.

ALGORITHM

```
def perform_genetic_algorithm(init_board,target_board, FITNESS_FUNCTION):


    iteration = 0
    while iteration < MAX_ITER and not task_completed(FITNESS_FUNCTION):
        if task_completed(FITNESS_FUNCTION):
          break
        cross_over()
        perform_mutations()
        population = next_gen_select(FITNESS_FUNCTION)
        iteration += 1

    if task_completed(FITNESS_FUNCTION):
      for p in population:
        if p.cost(FITNESS_FUNCTION) == 0:
          print("SUCCESS")
          print("Answer found in "+str(iteration+1)+" generations")
          print("Seq of actions required: ", p.genes)
          print("STEPS:")
          board = p.board
          printStringMatrix('Initial Board', board)
          for i in p.genes:
            board = move(board, i)
            printStringMatrix('Moving '+i, board)
          break
    else:
      print("FAILURE")
```

2) Compare and contrast (with justification) the results obtained from the two different Fitness functions. Justify the fitness function choice. Why was it chosen and how is it handling the problem in hand?

The fitness function is a function which takes a candidate solution to the problem as input and produces as output how "fit" or how "good" the solution is with respect to the problem in consideration. Calculation of fitness value is done repeatedly in a GA and therefore it should be sufficiently fast. A slow computation of the fitness value can adversely affect a GA and make it exceptionally slow.
A fitness function should possess the following characteristics –

- The fitness function should be sufficiently fast to compute.
- It must quantitatively measure how fit a given solution is or how fit individuals can be produced from the given solution.

Of the given fitness functions chosen, h1 is less computationally heavy and hence sufficiently fast as compared to h2 fitness function. For the purpose of this assignment, h1 and h2 cost

functions are chosen because they uniquely represent the cost of each state with respect to the target state

3) Comparisons b/w Simulated Annealing and Genetic Algorithm over 8 puzzle problem

TEST 1

Simulated Annealing

```
Enter the source_state: 12345678B
Enter the target_state: 123456B78
Source State
[['1' '2' '3']
 ['4' '5' '6']
 ['7' '8' 'B']]
Target State
[['1' '2' '3']
 ['4' '5' '6']
 ['B' '7' '8']]
```

Algorithm	Cooling Function	Path Cost	Final state reached	Path States	Execution Time	Reacha
h1(n)	1	2	123456B78	3	0.000107527	True
h1(n)	2	950	76B453182	951	0.319491	False
h1(n)	3	2	123456B78	3	5.55515e-05	True
h2(n)	1	1044	B23146758	1045	0.409699	False
h2(n)	2	266	B14827635	267	0.303221	False
h2(n)	3	40	81B752643	41	0.31401	False

Genetic Algorithm

```
H1 Heuristic Result:
SUCCESS
Answer found in 3 generations
Seq of actions required:  ['l', 'l']
STEPS:
Initial Board
[['1' '2' '3']
 ['4' '5' '6']
 ['7' '8' '0']]
Moving l
[['1' '2' '3']
 ['4' '5' '6']
 ['7' '0' '8']]
Moving l
[['1' '2' '3']
 ['4' '5' '6']
 ['0' '7' '8']]
```

```
H2 Heuristic Result:
SUCCESS
Answer found in 4 generations
Seq of actions required:  ['l', 'l']
STEPS:
Initial Board
[['1' '2' '3']
 ['4' '5' '6']
 ['7' '8' '0']]
Moving l
[['1' '2' '3']
 ['4' '5' '6']
 ['7' '0' '8']]
Moving l
[['1' '2' '3']
 ['4' '5' '6']
 ['0' '7' '8']]
```

TEST 2

Simulated Annealing

```
Enter the source_state: 413726058
Enter the target_state: 123456780
Source State
[['4' '1' '3']
 ['7' '2' '6']
 ['0' '5' '8']]
Target State
[['1' '2' '3']
 ['4' '5' '6']
 ['7' '8' '0']]
```

Algorithm	Cooling Function	Path Cost	Final state reached	Path States	Execution Time	Reachab
h1(n)	1	1173	B72546130413726058	1174	0.352477	False
h1(n)	2	2231	523764B50413726018	2232	0.327111	False
h1(n)	3	181	534027B61413726058	182	0.306126	False
h2(n)	1	129	250714B36413726058	130	0.299638	False
h2(n)	2	359	2546701B3413726058	360	0.303667	False
h2(n)	3	547	55367102B413726408	548	0.320839	False

Genetic Algorithm

```
H1 Heuristic Result:
SUCCESS
Answer found in 7 generations
Seq of actions required:  ['u', 'u', 'l', 'u', 'r', 'd', 'd', 'r']
STEPS:
```

```
H2 Heuristic Result:
SUCCESS
Answer found in 7 generations
Seq of actions required:  ['u', 'l', 'u', 'd', 'u', 'r', 'd', 'd', 'r']
STEPS:
Initial Board
```

TEST 3

Simulated Annealing

```
Enter the source_state: B13425786
Enter the target_state: 12345678B
Source State
[['B' '1' '3']
 ['4' '2' '5']
 ['7' '8' '6']]
Target State
[['1' '2' '3']
 ['4' '5' '6']
 ['7' '8' 'B']]
```

Algorithm	Cooling Function	Path Cost	Final state reached	Path States	Execution Time	Reachable	P
h1(n)	1	660	B68314725	661	0.346051	False	B
h1(n)	2	1428	53B628741	1429	0.315617	False	B
h1(n)	3	760	25B467813	761	0.311678	False	B
h2(n)	1	4	12345678B	5	0.000108242	True	B
h2(n)	2	1256	B62475183	1257	0.299844	False	B
h2(n)	3	4	12345678B	5	0.000113964	True	B

Genetic

```
H1 Heuristic Result:
SUCCESS
Answer found in 4 generations
Seq of actions required:  ['u', 'r', 'd', 'r', 'd']
STEPS:
```

```
H2 Heuristic Result:
SUCCESS
Answer found in 6 generations
Seq of actions required:  ['r', 'd', 'r', 'd']
STEPS:
Initial Board
[['0' '1' '3']
 ['4' '2' '5']
 ['7' '8' '6']]
Moving r
```

Analyze the results obtained with proper justifications. Describe your results on both algorithms and state the reasons for the difference of approach in both algorithms. Describe your views on what algorithm should have performed better for this particular problem and does your intuition match the results?

The following was observed from analysis:

- In terms of memory usage, Genetic Algorithm uses much overhead in storing the population elements as compared to Simulated Annealing
- In terms of time complexity, genetic algorithm performs faster than SA because it terminates quickly taking large steps during the start at multiple fronts
- SA may not give optimal path quickly and revert from higher fitness function by choosing randomly. The selection of these random successors is dependent on the cooling function and the cost function
- SA works better when we run it till infinity
- Genetic Algorithm gives a large sample space quickly

For this problem, where memory is not a constraint, Genetic Algorithm can be used as it can converge to a solution faster and give complete solutions

e. Is it infeasible to use Genetic Algorithm for an 8-Puzzle problem? If yes, Justify

Genetic Algorithm attempts to find the state closest to the target state in the population. It will be infeasible to use the Genetic algorithm for the 8 puzzle problem when the target is not at all reachable from the source state. The genetic algorithm will get stuck in an infinite loop. To avoid this, MAX_ITERATIONS = 5000 is used, meaning only 5000 generations of successors will be generated.