Problem Solving
Master 1, Visual Computing
2023/2024

Assignment 4
Solving the Knight's Tour problem using a genetic algorithm

The knight in chess is the piece that is shaped like a horse's head. The knight can move in an "L-shaped" pattern, advancing two squares in one direction and then having the option to turn either right or left. If we display all the current possible moves for the knight, the resulting pattern is as follows (see Figure 1).

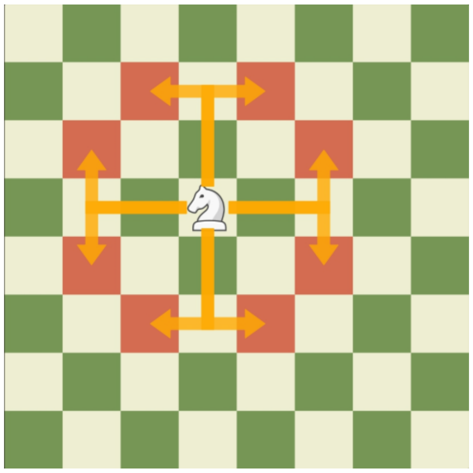


Figure 1 All knight's possible moves

The knight can move to the square marked in red and then repeat the process on the new square. A knight's tour is a sequence of moves that a knight can make to visit every chess square exactly once.

The objective of this assignment is to tackle the knight's tour problem using a genetic algorithm. For this purpose, we will create the following classes:

1. **Chromosome class:** In the knight's tour problem, we will use the moves made by the knight as the genes of its chromosome. we define the class **Chromosome** with the following attributes:

✧ **genes:** an array of length 63 representing the knight's moves. Each gene represents one of the 8 possible moves.

In the Chromosome class, you should also define the following functions:

✧ **init (genes):** The function that creates a new chromosome. If no genes are given (as in the case of chromosomes in the initial population), the chromosome will generate a random set of genes.

✧ **crossover (partner):** The crossover function takes another chromosome as input and combines genes from both chromosomes to form the new offspring. For this problem, we will use the single-point crossover.

- ✧ **mutation ( ):** The mutation introduces a probability for genes to mutate, implying that they will transform into a random move. This feature motivates the Genetic Algorithm to explore new moves.

2. **Knight class:** Each knight will store these variables:

- ✧ **position:** the coordinates (x, y) for the knight's current position
- ✧ **chromosome:** the sequence of moves taken by the knight.
- ✧ **path:** the list of knight's positions after applying the moves defined in the chromosome.
- ✧ **fitness:** the value of the fitness function. The maximum value is 64, representing the number of squares on the chessboard.

In the Knight class, the following functions should be defined:

- ✧ **init (chromosome):** This function creates a new knight. If no chromosome is given (as in the case of knights in the initial population), the knight will generate a new chromosome. Additionally, this function sets the current position of the knight to (0, 0), the fitness value to 0, and saves the initial position into the path list.
- ✧ **move_forward (direction):** This function moves the knight in one of the 8 directions (***1: up-right, 2: right-up, 3: right-down, 4: down-right, 5: down-left, 6: left-down, 7: left-up, 8: up-left***). It generates the new position of the knight after applying a move.
- ✧ **move_backward (direction):** This function allows the knight to trace back if the applied move is illegal.
- ✧ **check_moves ( ):** This function checks the validity of each move in the chromosome array. A move is considered invalid if, when applied using the function **move_forward (direction)**, it places the knight outside the chessboard or in a position that has already been visited. If a move is illegal, this function corrects it by canceling the move using **move_backward (direction)**. It then tests other moves by cycling forward or backward. The direction of the cycle is determined randomly and remains the same for the entire set of moves in the chromosome. If no valid move is found, the last move is retained. For example, if the current move is ***4: down-right*** and is illegal, and a cycle forward is chosen, the moves will be selected in the order: ***5: down-left, 6: left-down, 7: left-up, 8: up-left, 1: up-right, 2: right-up, 3: right-down***. If a cycle backward is chosen, the moves will be selected in the order: ***3: right-down, 2: right-up, 1: up-right, 8: up-left, 7: left-up, 6: left-down, 5: down-left***.
- ✧ **evaluate_fitness ( ):** This function loops through the knight's path (the list of the knight's visited positions) and increments the fitness value by one until an invalid move is encountered. If the knight has visited all squares on the chessboard, the fitness value is equal to 64.

3. **Population Class:** The population class will consist of the following attributes:

- ✧ **population_size:** The size of the population, e.g., 50.
- ✧ **generation:** The number of generations, initially set to 1.
- ✧ **knights:** A population is composed of a list of knights.

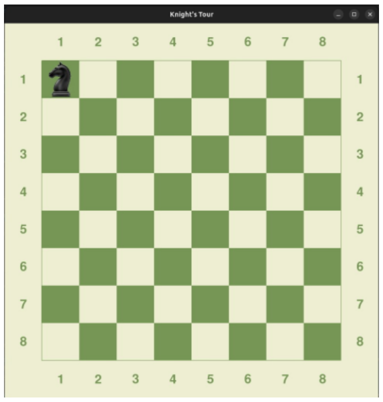
In the **Population class**, the following functions should be defined:

- ✧ **init(population_size):** This function generates the list of knights for the initial population. It also initializes the number of generations to 1.
- ✧ **check_population():** This function loops through the knights of the population and checks the validity of the moves of each knight using the function **check_moves()** defined in the **Knight class**.
- ✧ **evaluate():** This function evaluates the fitness of every individual/knight in the population using the function **evaluate_fitness()** defined in the **Knight class** and returns the best knight with its fitness.

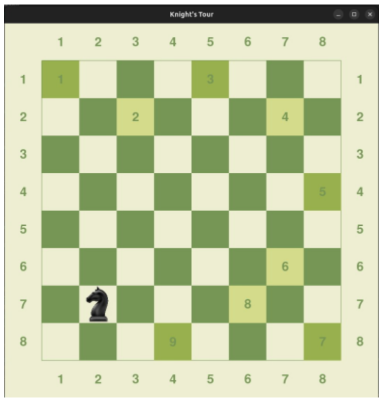
- **tournament_selection(size):** To select parental combinations for crossover, tournament selection is employed with a sample size of *n*. In tournament selection, *n* knights are randomly sampled from the population, and a tournament is conducted on these samples by choosing the two best ones based on their fitness values. A sample size of 3 is taken.
- **create_new_generation():** This function creates a population for the new generation with the same size as the current population. In each iteration, two parents are selected using the function **tournament_selection(size)**. The crossover between the parental combination is then applied to generate two new offspring using the function **crossover(partner)** defined in the **Chromosome class**. Afterward, mutation is applied to each offspring using the function **mutation()** defined in the **Chromosome class**. This function also increments the number of generations.

5. **The main function:** The main function applies a run of the genetic algorithm and displays the optimal solution on an interface, as illustrated in Figure 2.

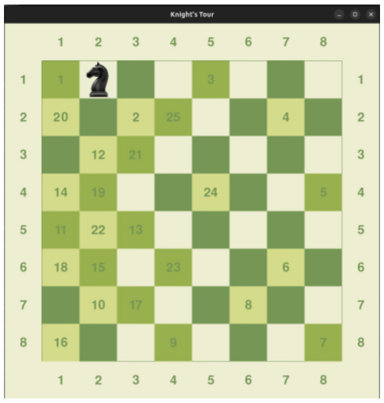
```
def main ( ):
   population_size = 50
   # Create the initial population
   population = Population(population_size)

   while True:
      # Check the validity of the current population
      population.check_population()

      # Evaluate the current generation and get the best knight with its fitness value
      maxFit, bestSolution = population.evaluate ( )
      if maxFit == 64:
         break

      # Generate the new population
      population.create_new_generation ()

   # Create the user interface to display the solution

if __name__ == "__main__":
   main()
```



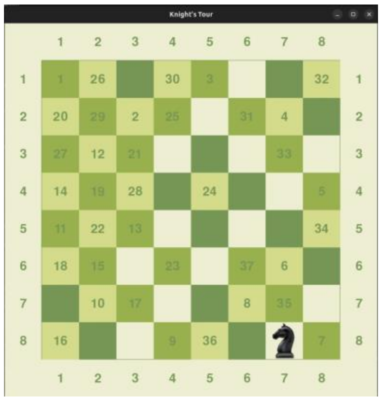
(a)



(b)

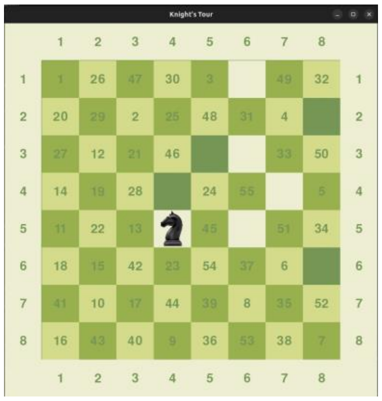


(c)



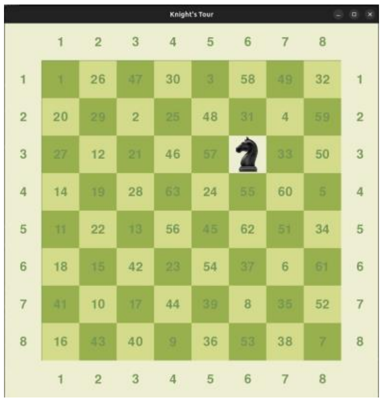
	1	2	3	4	5	6	7	8	
1	1	26		30	3			32	1
2	20	29	2	25		31	4		2
3	27	12	21				33		3
4	14	19	28		24			5	4
5	11	22	13					34	5
6	18	15		23		37	6		6
7		10	17			8	35		7
8	16			9	36			7	8
	1	2	3	4	5	6	7	8	

(d)



	1	2	3	4	5	6	7	8	
1	1	26	47	30	3		49	32	1
2	20	29	2	25	48	31	4		2
3	27	12	21	46			33	50	3
4	14	19	28		24	55		5	4
5	11	22	13		45		51	34	5
6	18	15	42	23	54	37	6		6
7	41	10	17	44	39	8	35	52	7
8	16	43	40	9	36	53	38	7	8
	1	2	3	4	5	6	7	8	

(e)



	1	2	3	4	5	6	7	8	
1	1	26	47	30	3	58	49	32	1
2	20	29	2	25	48	31	4	59	2
3	27	12	21	46	57		33	50	3
4	14	19	28	63	24	55	60	5	4
5	11	22	13	56	45	62	51	34	5
6	18	15	42	23	54	37	6	61	6
7	41	10	17	44	39	8	35	52	7
8	16	43	40	9	36	53	38	7	8
	1	2	3	4	5	6	7	8	

(f)

Figure 2 visualization of a solution of the knight's tour problem using the genetic algorithm