Report on Project 1 IT3105: Implement and Compare the Performance of AI Algorithms to Solve The n-Queens Problem

a) Part-1:

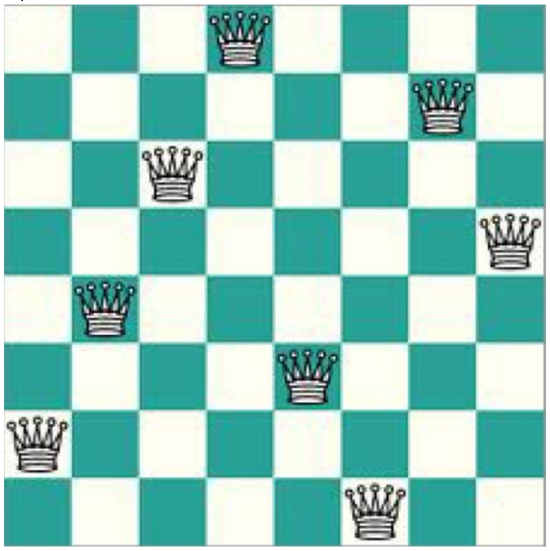
Figure 1: The position on this board is represented by the following vector [2, 4, 6, 8, 3, 1, 7, 5]

The board representation is a one-dimensional vector containing integers from 1 to n. Each queens column position is represented by its place, col, in the vector. The row position is represented by the integer value in place col.
This representation is easy to understand and at the same time it is compact. The fact that it can be traversed in linear time is the main reason for the chosen representation.

The heuristic function used returns the number of threats on a queen without counting duplicates (if Qa is threatened by Qb, Qb is not threatened by Qa). A lower value, means we are in a better state. A value equal to zero means the state is a solution. The way the heuristic is calculated is by looking at one column at the time. For each column, all earlier columns are evaluated. If a threat is detected the threat count is incremented. When all columns are evaluated the threat count is returned as the heuristic value of the current state.

b) Part-2:

for input: 2 4 8 14 0 0 0 0 0 0 … n

n	Solutions found	Time taken avg 10 runs(ms)	Fastest time (ms)	Slowest time (ms)
14	10	38	35	43
16	456	134	129	143
18	9449	1440	1396	1460
20	312652	24749	24556	24960

c) Part-3:

There will be used two inputs. One random and one with the maximum amount of conflicts

N:	18
Input1.1:	2 4 8 1 5 14 16 9 1 10 7 2 4 14 17 11 9 13
Input1.2:	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

N:	20
Input2.1:	9 13 20 16 11 13 7 9 2 1 6 15 19 4 8 16 10 3 19 5
Input2.2:	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

N:	24
Input3.1:	20 1 4 18 14 22 15 11 2 8 5 12 24 16 14 3 6 4 20 2 17 23 13 8
Input3.2:	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

N:	30
Input4.1:	1 27 4 7 10 14 17 22 29 20 13 8 6 15 2 25 30 14 11 19 7 6 4 1 15 25 28 24 12 6
Input4.2:	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Tabu Search Algorithm

Pseudocode:

```
Board = initial input
While(termination condition not met) do
        If(checkForSolution(board))
                Return board
        neighbours = generateNeighbours(board) - tabulist
        bestNeighbour = findFittestNeighbour(neighbours)
        UpdateTabuList(bestNeighbour)
        Board = bestNeighbour
Code end
```

Notes:

The tabu list is represented by the changes made to the board, rather than the entire boards. This makes the algorithm more space efficient.

The fitness function used to evaluate different states is the same one mentioned in part 1 of this paper.

Simulated Annealing Algorithm

Pseudocode:

```
Board = initial input
Temperature = startTemp
While (termination conditions not met) do
    If(checkForSolution(board))
        Return board
    nextBoard = generateNeighbour(board)
    if(value(nextBoard) <= value(board))
        board = nextBoard
    else
        delta = value(nextBoard)  - value(board))
        probability = $e^{-(delta/T)}$
        x = randomDouble(0,1)
        if (probability > x)
            board = nextBoard
    T = T * temp_scale
Code end
```

The fitness function used for evaluating the board values is the same mention in part-1 of this paper.

Genetic Algorithm

The fitness function used is the same as the heuristic described in part-1 of this paper.

GA design:

- Chromosome representation: The representation is a vector of length n. Each entry represents a column on the board. The value of each entry range from 1 to n (included) and represents the row the queen in that column is placed.

- Implementation of crossover: The crossover used is Order1-crossover and it ensures unique rows for all queens in the child state. It takes a random portion of parent1 and copies it to the child. Then it takes the first remaining queens, from rows that are not present in parent1, from parent2 and copies them to the child.

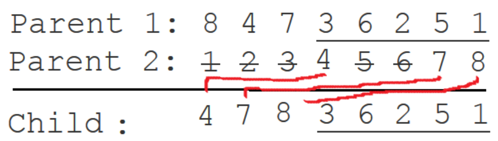
Parent 1: 8 4 7 3 6 2 5 1
Parent 2: ~~1~~ ~~2~~ ~~3~~ 4 ~~5~~ ~~6~~ 7 8
Child : 4 7 8 3 6 2 5 1

Figure 2: Child created by using Order1-crossover on parent1 and parent2

- Mutation operators: The mutation operator flips two columns. The queen in col1 is assigned the value row2 and the queen in col2 is assigned the value row1.
- Selection strategy: When selecting parents the chance of selection an individual is proportional to the fitness value of the individual. The value of the randomly selected individual is divided by the total population value and scaled, with some constant, to punish worse individuals.

The algorithm will produce infeasible offspring as only solution states are feasible. However, the algorithm manages to reduce the production of the most obvious infeasible states by ensuring that every queen is placed in a unique column and row. This removes horizontal and vertical collisions and leaves only the diagonal collisions.

Performance:

The algorithm finds the most solutions in shortest time with the given parameters:

Population size	number of generations before restart	Crossover rate	Mutation rate
5	7000	0.5	0.25

Comparison:

Board size(n) - input	No. of sol for TS	Time to calc TS (s)	No. of sol for SA	Time to calc for SA (s)	No. of sol for GA	Time to calc for GA (s)
18 – 1.1	10000	16	10000	22	5000	74
18 - 1.2	10000	15	10000	21	5000	73
20 – 2.1	10000	17	10000	26	5000	87
20 – 2.2	10000	19	10000	25	5000	93
24 – 3.1	10000	27	10000	38	5000	125
24 – 3.2	10000	30	10000	38	5000	129
30 – 4.1	10000	50	10000	62	5000	208
30 – 4.2	10000	51	10000	67	5000	213