# Assignment 1

*Personal information deleted*

## 1

I found your paper on the N-queen problem very instructive, but to find a better function for fitness measuring, I prefer to revise it a little bit.

For the original method, the point is to measure how many queens are attacked. Enough to detect whether the problem is solved though, it can't tell us which situation is worse with the same number of queens being attacked.

The proposed method calculates how many attacks are there on the board instead. Using the same pattern expression $Q$, for each queen we number them by column from 0 to 7 as $q_i$. By calculating how many times a queen is attacked, we can have an objective function for a specific queen.

$$\mathrm{f}(q_i) = \sum_{j=i+1}^{8} g(q_i, q_j)$$

where

$$\mathrm{g}(q_i, q_j) = \begin{cases} 1, if\, q_i = q_j or |q_i - q_j| = |i - j| \\ 0, otherwise \end{cases}$$

In this method, $f(q_i)$ indicates for a queen in a specific column, how many queens it attacks, while $g(q_i,q_j)$ indicates whether it attacks the queen in the j-th column.

By summing them up, we will thus have the fitness function for the 8-queen problem:

$$\mathrm{F}(Q) = \sum_{i=1}^{7} f(q_i)$$

The smaller $F(Q)$ the better queen distribution.

Note that we don't need to calculate how many attacks the 8-th performs, because we consider the attack as an intrinsically mutual relationship, and this is also why we only consider $g(q_i,q_j)$ from j = i+1.

## 2

For an 8-queen problem in 64-bit expression, the main idea remains the same that we start from calculating how many times a queen successfully attacks and sum them up.

One obvious solution is to convert the 64-bit expression into the expression in Q1.

For a 64-bit Q expression, a queen takes up the n-th position when Q[n]=1, while n is an integer within the range of 0 to 63. A conversion relationship for the i-th queen can be easily found as

$$p_i = (x_i, y_i) = \left(\left[\frac{n}{8}\right] + 1, n\%8\right)$$

In the given case, two queens are possibly in a same column, therefore,

$$\mathrm{f}(q_k) = \sum_{j=k+1}^{8} g(p_i, p_j)$$

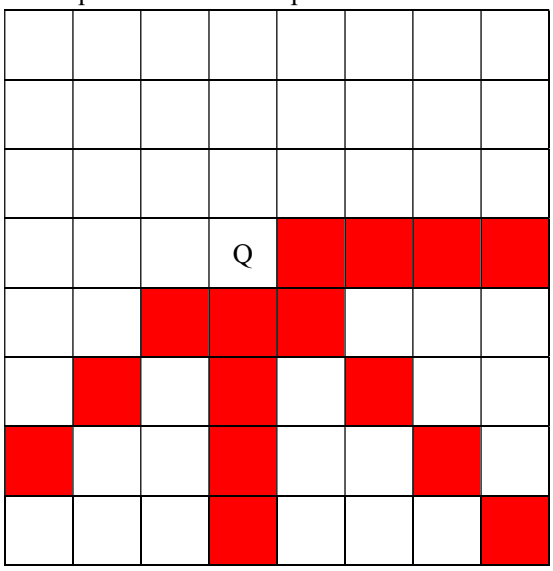
where

$$\mathrm{g}(p_i, p_j) = \begin{cases} 1, if\, x_i = x_j or y_i = y_j or |x_i - x_j| = |y_i - y_j| \\ 0, otherwise \end{cases}$$

and thus

$$\mathrm{F}(Q) = \sum_{i=1}^{7} f(p_i)$$

Another solution is derived directly from the 64-bit pattern. Still, we consider the attack as a mutual relationship, so only half of a queen's attack direction is considered: right, lower-right, down, lower-left, and of course locations of queens don't overlap.




Given its complexity, the readability will be very poor if we go all the way down to have a specific mathematic expression for the function, so it is therefore omitted; however, pseudo-code of the algorithm can be given clearly, with a 64-bit $Q$ as the input.

```
// Find every position of queens
for (n = 0 to 63) loop:
  i = 0
  if Q[n] = 1:
```

```
    q[i] = n
    i = i+1
  endif
  n = n+1
endloop

// Initialize F(Q)
F(Q) = 0
// Detect attack targets
for (i = 0 to 6) loop:
  f(qi) = 0
  for (j = i+1 to 7) loop:
    //right attack
    if ((q[i] < q[j] && q[j] <= (floor(q[i]/8)+1)*8)):
      f(qi) = f(qi)+1
    endif
    for (row = (floor(q[i]/8)+1) to 8) loop:
      //lower-right, down, lower-left attacks
      if (q[j] = q[i]+row*9 || q[j] = q[i]+row*8 || q[j] = q[i]+row*7)
        f(qi) = f(qi)+1
      endif
    endloop
  endloop

  // Sum up
  F(Q) = F(Q)+f(qi)
endloop
```

The algorithm is quite straightforward. I believe optimization exists for this algorithm (e.g., object orientated programming) but let's just leave it for now.