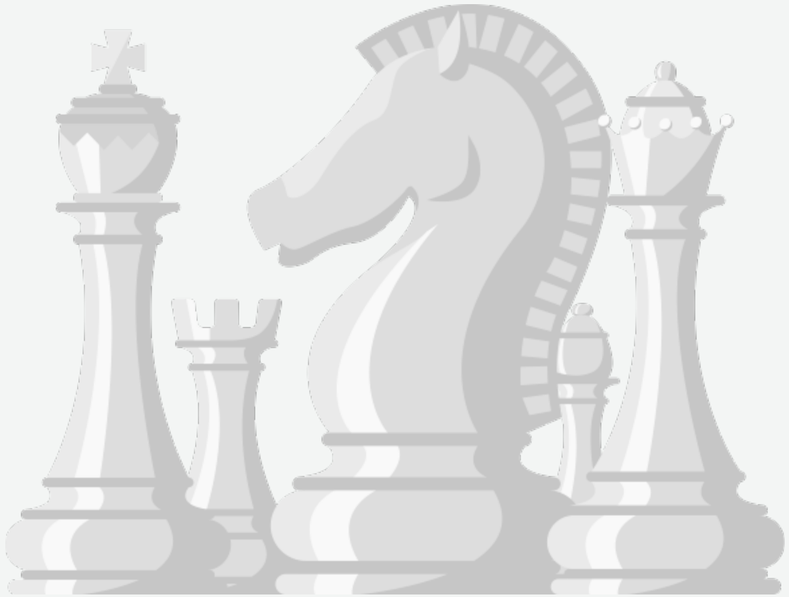


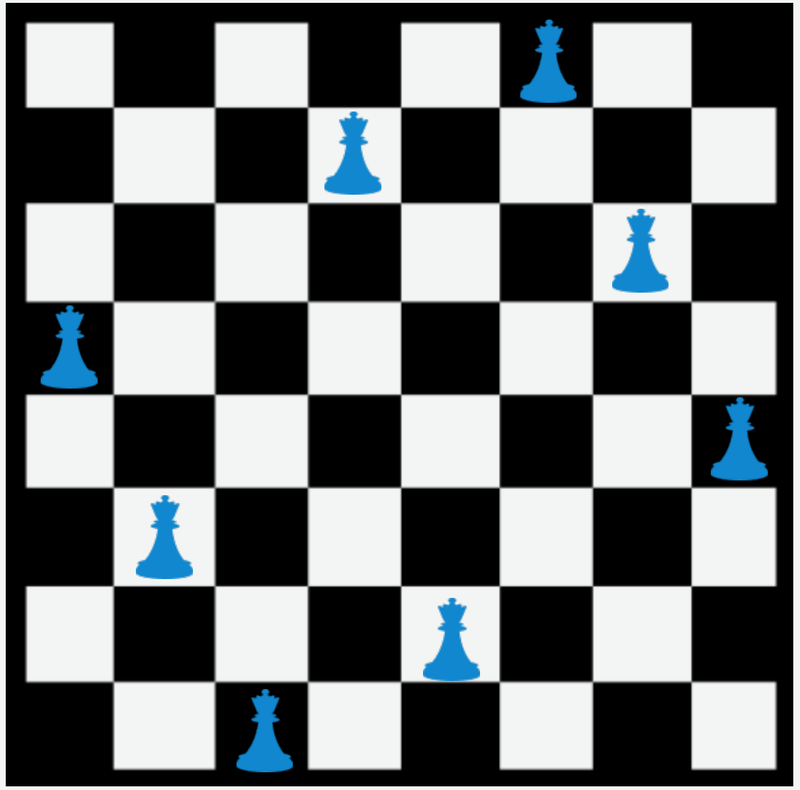
Solving 8 - Queens Problem using Genetic Algorithm

Statistical Computation Final Project

212STG01 강지영
212STG07 노기정
212STG21 이선경

INDEX

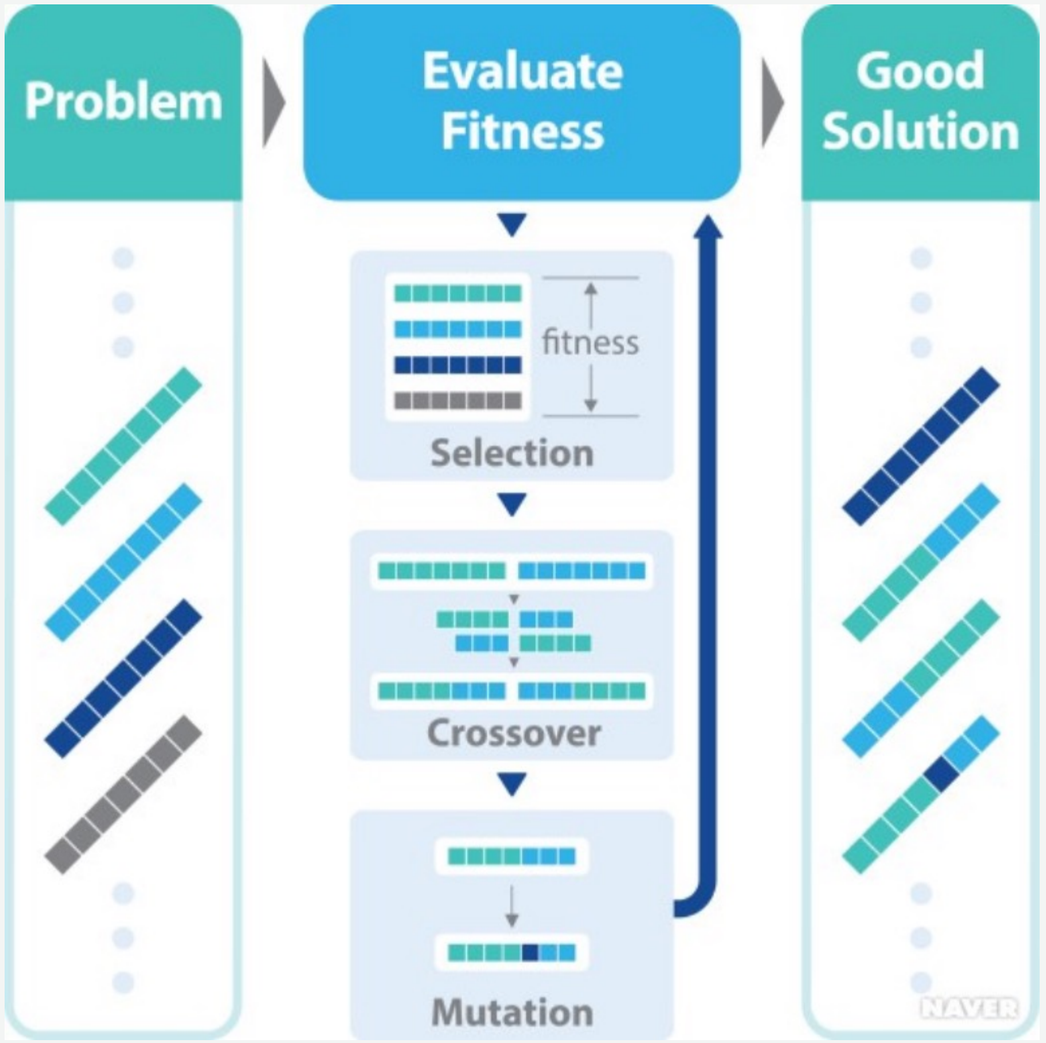
N- Queens Problem: nxn chessboard에 서로 공격하지 않는 queen을 놓는 방법
-> nxn chessboard에 서로 공격하지 않는 queen 최대 수: n개



8 queens

n queens	solution
1	1
2	0
3	0
4	2
5	10
6	4
7	40
8	92

알고리즘 설명 | Genetic Algorithm Procedure 설명



Step 1 : Generate Initial Random solutions

n개의 랜덤한 chromosome(염색체) 생성

Step 2 : Evaluate the fitness of chromosome

각 chromosome의 fitness(적합도) 계산

Step 3 : Select chromosomes as parents

fitness를 기반으로 부모가 될 chromosome들을 확률적으로 결정

Step 4 : Crossover of parent chromosomes

부모 chromosome을 여러 방법으로 결합해서 offspring(자손) 생성

Step 5 : Mutation of offspring chromosomes

offspring이 가진 chromosome 일부를 랜덤하게 변형
Mutation을 통해 local optima에 빠질 위험 감소

문제해결과정 | Solving N - Queens Problem using Genetic Algorithm 논문 리뷰

8x8 체스보드에 서로 공격하지 않는 Queen을 두는 방법 찾기

8x8일 때 서로 공격하지 않는 Queen의 개수는 최대 8개
n=8일 때의 가능한 solution의 수는 92개가 되고,
Genetic Algorithm 을 적용해 92가지의 solution을 찾는다.

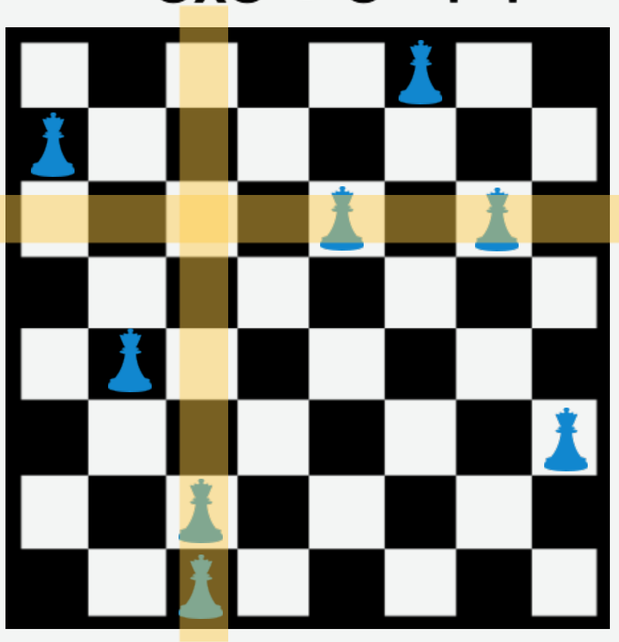
Solving N Queen Problem using Genetic Algorithm

July 2015 · International Journal of Computer Applications 122(12):11-14
DOI:10.5120/21750-5005

Authors:

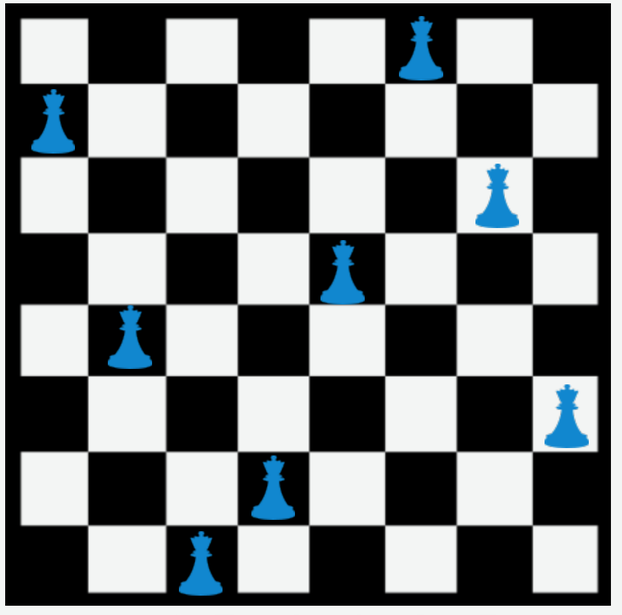
Ahmed mohamed Farhan
Imam Abdul Rahman bin Faisal University

$_{8x8}C_8$ 가지



서로 같은 행이나 열에 있으면 무조건 공격

8! 가지

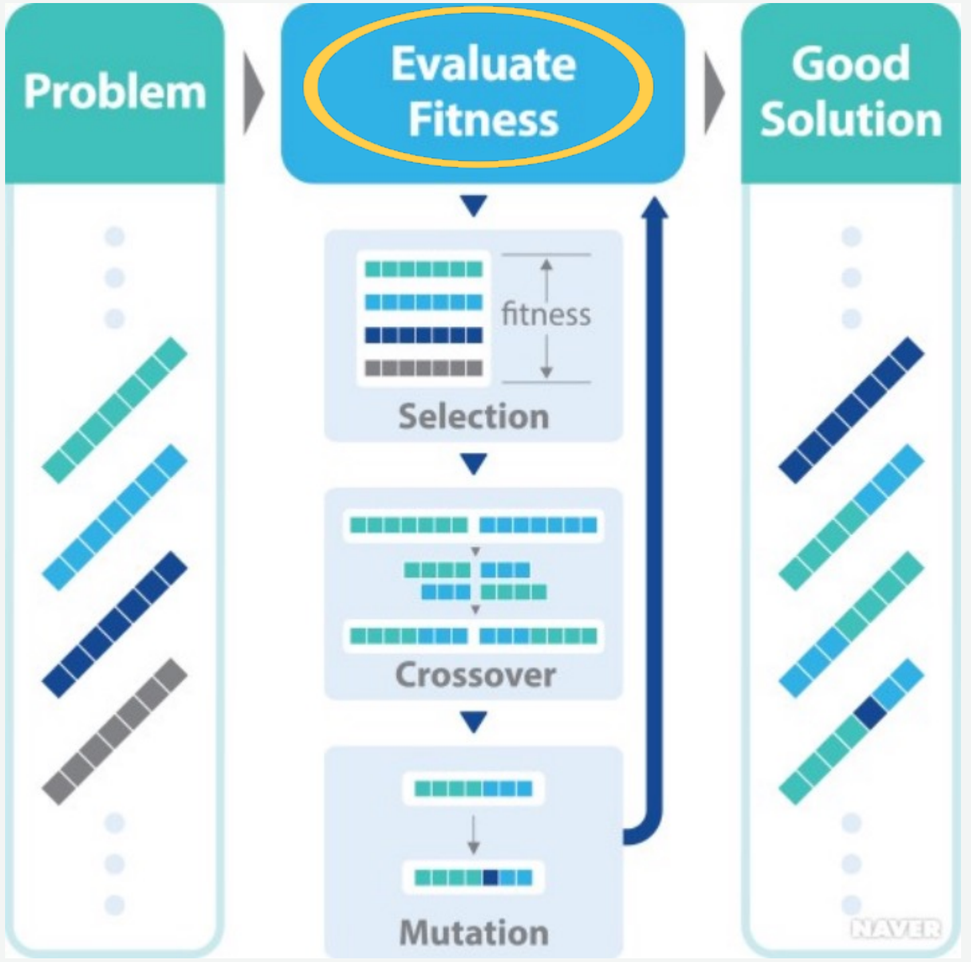


Chromosome = (7 4 1 2 5 8 6 3)

→ 가능한 경우의 수와 해는 찾는 시간이 줄어들게 된다.

문제해결과정 | Solving N - Queens Problem using Genetic Algorithm 논문 리뷰

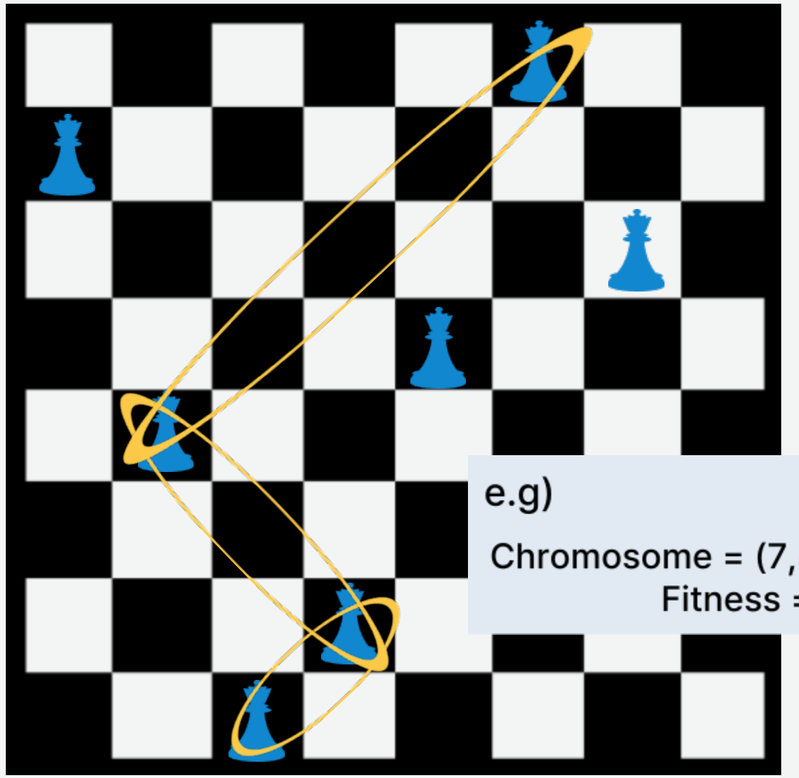
Fitness Function



서로 공격하는 (같은 대각선에 존재하는) 퀸들의 쌍의 수 = x

Fitness fuction F(x) = 1 / (x+1)

(max Fitness = 1)인 Chromosome = Solution



e.g)

Chromosome = (7,4,1,2,5,8,6,3) 일 때

Fitness = 1/(1+3)

(7 4 1 2 5 8 6 3)

Fitness Function

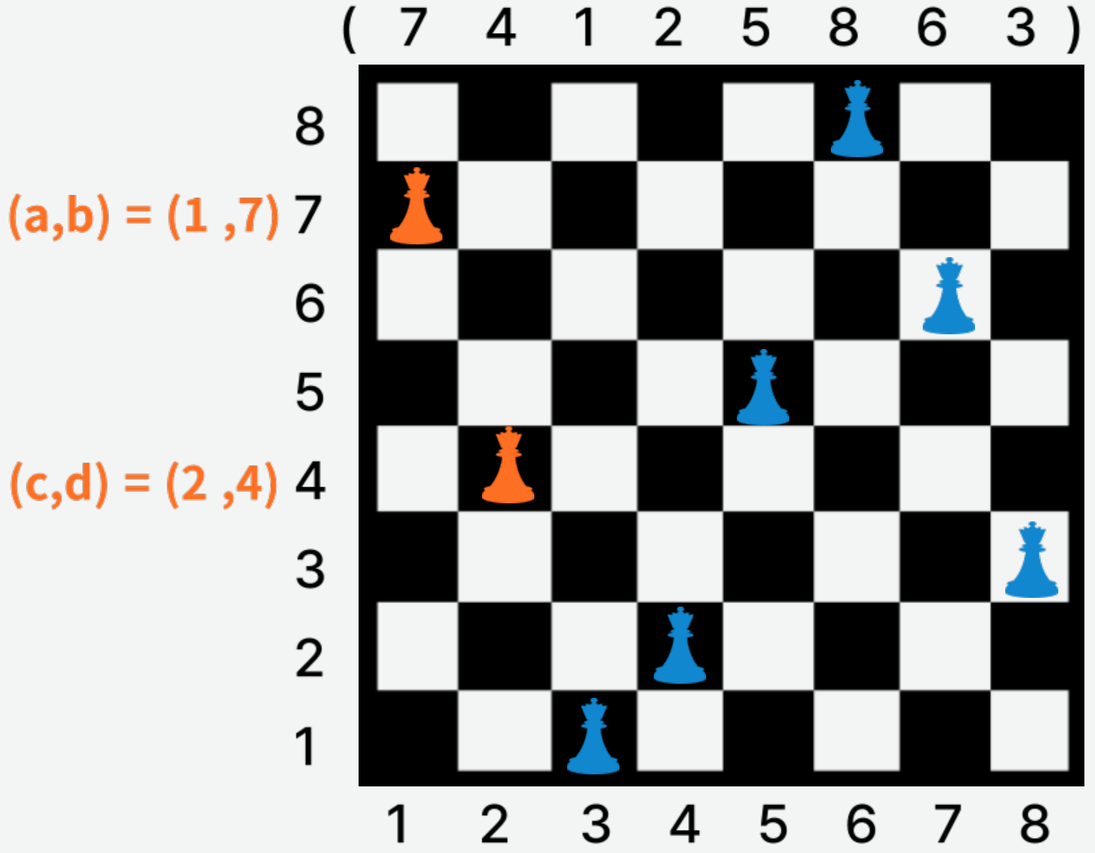
- 서로 공격하는 (같은 대각선에 존재하는) 퀸들의 쌍의 수 계산

모든 a, c에 대해

| a - c | == | b - d | 인지 확인 맞으면 1 아니면 0
대각선 상에 존재하려면 x좌표의 차이 = y좌표의 차이

| 1 - 2 | =/= | 7 - 4 |

* a, c에 대한 b,d는 chromosome에 의해 결정



Fitness Function

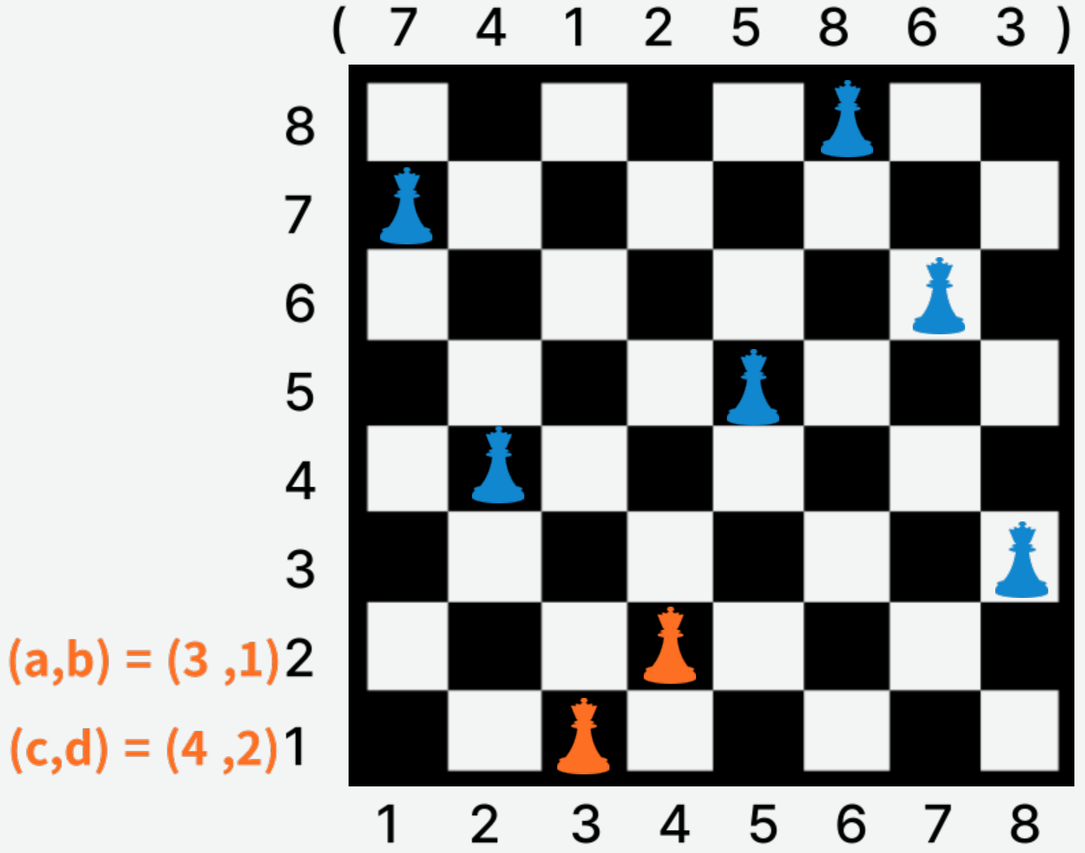
- 서로 공격하는 (같은 대각선에 존재하는) 퀸들의 쌍의 수 계산

모든 a, c에 대해

| a - c | == | b - d | 인지 확인 맞으면 1 아니면 0
대각선 상에 존재하려면 x좌표의 차이 = y좌표의 차이

| 3 - 4 | == | 1 - 2 |

* a, c에 대한 b,d는 chromosome에 의해 결정



문제해결과정 | Solving N - Queens Problem using Genetic Algorithm 논문 리뷰

논문에서 제시한 내용으로 구현 (3 - Way 토너먼트)

Intial population

[[1]]
1 3 4 2 7 5 8 6
[[2]]
7 3 5 1 6 4 2 8
[[3]]
7 8 3 2 6 5 1 4
[[4]]
8 7 6 1 2 4 3 5
[[5]]
7 8 6 5 4 1 2 3
[[6]]
4 6 7 2 8 3 5 1
.
.
.
[[91]]
2 1 5 6 7 4 8 3
[[92]]
5 6 4 1 2 7 3 8

Intial population

- 92개의 RANDOM Chromosome 생성

Selection

3- WAY TOURNAMENT

Crossover

1. OX CROSSOVER
2. PMX CROSSOVER
3. 3 - WAY TOURNAMENT CROSSOVER

Mutation

- 두 부분을 랜덤으로 골라 위치를 서로 교환

Replacement

- 자식으로 가장 fitness가 작았던 개체를 대체

Fitness = 1 인 개체가 나올 때 까지 Crossover, Mutation, Replacement 반복

문제해결과정 | Solving N - Queens Problem using Genetic Algorithm 논문 리뷰

논문에서 제시한 내용으로 구현 (3 - Way 토너먼트)

Random 3- Pick

Intial population

- 92개의 RANDOM Chromosome 생성

Selection

3- WAY TOURNAMENT
population 에서 랜덤하게 3개의 개체를 골라
Fitness가 큰 순으로 Rank,
3개의 개체(chromosome)중 Fitness가 큰 2마리의
개체가 부모가 된다.

Crossover

1. OX CROSSOVER
2. PMX CROSSOVER
3. 3 - WAY TOURNAMENT CROSSOVER

Mutation

- 두 부분을 랜덤으로 골라 위치를 서로 교환

Replacement

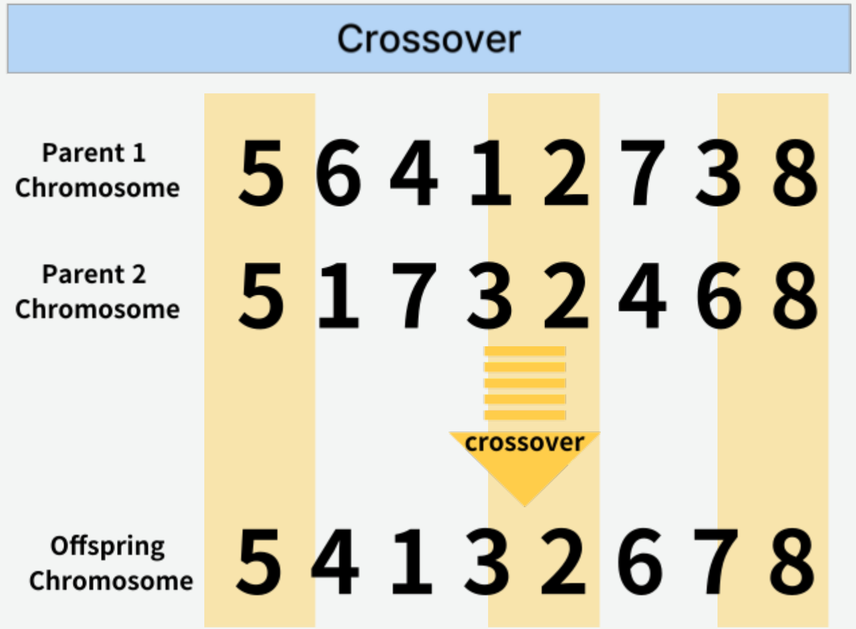
- 자식으로 가장 fitness가 작았던 개체를 대체

Fitness = 1 인 개체가 나올 때 까지 Crossover, Mutation, Replacement 반복

문제해결과정 | Solving N - Queens Problem using Genetic Algorithm 논문 리뷰

논문에서 제시한 내용으로 구현 (3 - Way 토너먼트)

[[92]]
1 5 6 4 1 2 7 3 8
[[72]]
2 5 1 7 3 2 4 6 8
[[5]]
3 7 8 6 5 4 1 2 3



* 만약 두 Parents가 똑같은 chromosome이 나온다면 하나를 랜덤으로 다시 생성

Intial population

- 92개의 RANDOM Chromosome 생성

Selection

3- WAY TOURNAMENT

Crossover

1. OX CROSSOVER
2. PMX CROSSOVER
3. **3 - WAY TOURNAMENT CROSSOVER**

두 부모가 갖고있는 공통된 부분은 남기고
다른 부분은 무작위로 섞는
crossover를 통해 하나의 자식을 만든다.

Mutation

- 두 부분을 랜덤으로 골라 위치를 서로 교환

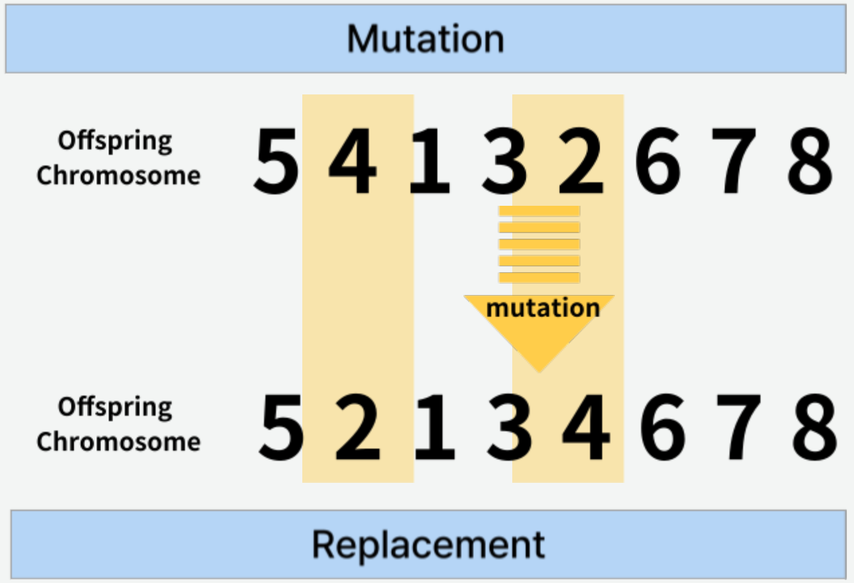
Replacement

- 자식으로 가장 fitness가 작았던 개체를 대체

Fitness = 1 인 개체가 나올 때 까지 Crossover, Mutation, Replacement 반복

문제해결과정 | Solving N - Queens Problem using Genetic Algorithm 논문 리뷰

논문에서 제시한 내용으로 구현 (3 - Way 토너먼트)



1	[[92]] 5 6 4 1 2 7 3 8
2	[[72]] 5 1 7 3 2 4 6 8
3	[[New]] 5 2 1 3 4 6 7 8

Intial population

- 92개의 RANDOM Chromosome 생성

Selection

3- WAY TOURNAMENT

Crossover

1. OX CROSSOVER
2. PMX CROSSOVER
3. 3 - WAY TOURNAMENT CROSSOVER

Mutation

- pmut ation = 0.1의 확률로
새로 생긴 자식 Chromosome의 두 부분을
랜덤으로 골라 위치를 서로 바꿔준다.

Replacement

- 자식으로 가장 fitness가 작았던 개체를 대체

Fitness = 1 인 개체가 나올 때 까지 Crossover, Mutation, Replacement 반복

문제해결과정 | Solving 8 - Queens Problem using GA - R package 이용

GA - R package 이용 : OX CROSSOVER

Intial population

(popSize)개의 랜덤 Chromosome 생성
각각의 fitness에 비례하게 생존확률 p 할당

	Fitness	p
[[1]] 1 3 4 2 7 5 8 6	0.11111111	p_1
[[2]] 7 3 5 1 6 4 2 8	0.14285714	p_2
[[3]] 7 8 3 2 6 5 1 4	0.11111111	p_3
[[4]] 2 5 8 4 7 3 1 6	0.33333333	p_4
.		
.		
[[popSize-1]] 2 1 5 6 7 4 8 3	0.07692308	$p_{(popSize-1)}$
[[popSize]] 5 6 4 1 2 7 3 8	0.11111111	$p_{(popSize)}$

Selection

부모 population에서 **생존확률 p를 weight**으로
2개의 RANDOM Chromosome 선택
with replacement

[[1]]
1 3 4 2 7 5 8 6
[[2]]
2 5 8 4 7 3 1 6
[[3]]
5 6 4 1 2 7 3 8
.
.
[[48]]
5 4 1 3 2 6 7 8
.
[[popSize-1]]
2 5 8 4 7 3 1 6
[[popSize]]
5 6 4 1 2 7 3 8

Crossover

< OX CROSSOVER >

Parent 1 Chromosome	5 6 | 4 1 2 | 7 3 8
Offspring 1 Chromosome	5 3 | 4 1 2 | 6 7 8
Parent 2 Chromosome	5 4 1 3 2 6 7 8
Offspring 2 Chromosome	5 6 1 3 2 4 7 8

Mutation

새로 생긴 자식 Chromosome의
두 부분을 랜덤으로 골라 위치를
서로 바꿔준다.

Offspring Chromosome 5 4 1 3 2 6 7 8
mutation
Offspring Chromosome 5 2 1 3 4 6 7 8

Replacement

새로운 (popSize)개의 population 생성
Fitness = 1 인 개체가 나올 때 까지 Selection, Crossover, Mutation 반복

문제해결과정 | Solving 8 - Queens Problem using GA - R package 이용

GA - R package 이용 : OX CROSSOVER

Intial population

(popSize)개의 랜덤 Chromosome 생성
각각의 fitness에 비례하게 생존확률 p 할당

	Fitness	p
[[1]] 1 3 4 2 7 5 8 6	0.11111111	p_1
[[2]] 7 3 5 1 6 4 2 8	0.14285714	p_2
[[3]] 7 8 3 2 6 5 1 4	0.11111111	p_3
[[4]] 2 5 8 4 7 3 1 6	0.33333333	p_4
.		
.		
[[popSize-1]] 2 1 5 6 7 4 8 3	0.07692308	$p_{(popSize-1)}$
[[popSize]] 5 6 4 1 2 7 3 8	0.11111111	$p_{(popSize)}$

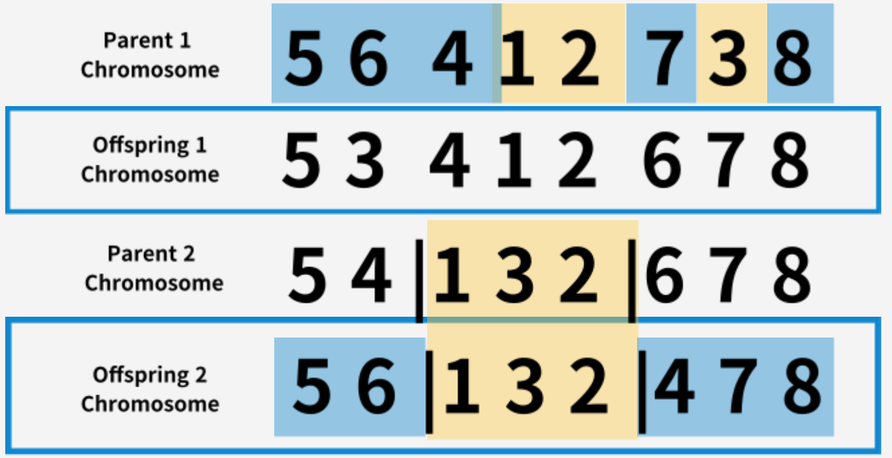
Selection

부모 population에서 **생존확률 p를 weight**으로
2개의 RANDOM Chromosome 선택
with replacement

[[1]]
1 3 4 2 7 5 8 6
[[2]]
2 5 8 4 7 3 1 6
[[3]]
5 6 4 1 2 7 3 8
.
.
[[48]]
5 4 1 3 2 6 7 8
.
[[popSize-1]]
2 5 8 4 7 3 1 6
[[popSize]]
5 6 4 1 2 7 3 8

Crossover

< OX CROSSOVER >



Mutation

새로 생긴 자식 Chromosome의
두 부분을 랜덤으로 골라 위치를
서로 바꿔준다.

Replacement

새로운 (popSize)개의 population 생성
Fitness = 1 인 개체가 나올 때 까지 Selection, Crossover, Mutation 반복

문제해결과정 | Solving 8 - Queens Problem using GA - R package 이용

GA - R package 이용 : PMX CROSSOVER

Intial population

(popSize)개의 랜덤 Chromosome 생성
각각의 fitness에 비례하게 생존확률 p 할당

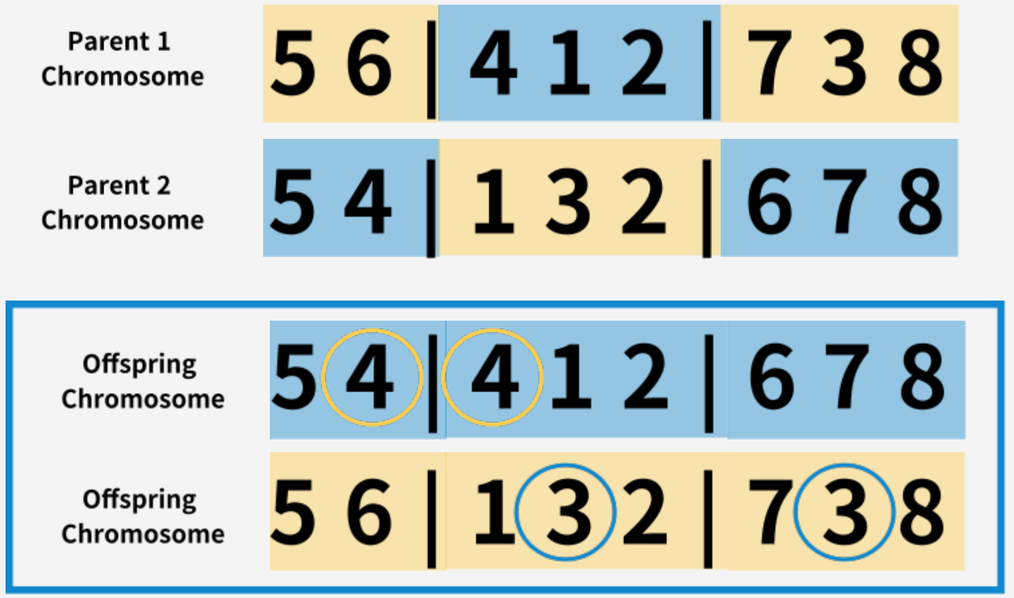
Selection

부모 population에서 **생존확률 p를 weight**으로
2개의 RANDOM Chromosome 선택
with replacement

Crossover

< PMX CROSSOVER >

STEP 1)



Mutation

새로 생긴 자식 Chromosome의 두 부분을 랜덤으로 골라 위치를 서로 바꿔준다.

Replacement

새로운 (popSize)개의 population 생성
Fitness = 1 인 개체가 나올 때 까지
Selection, Crossover, Mutation 반복

문제해결과정 | Solving 8 - Queens Problem using GA - R package 이용

GA - R package 이용 : PMX CROSSOVER

Intial population

(popSize)개의 랜덤 Chromosome 생성
각각의 fitness에 비례하게 생존확률 p 할당

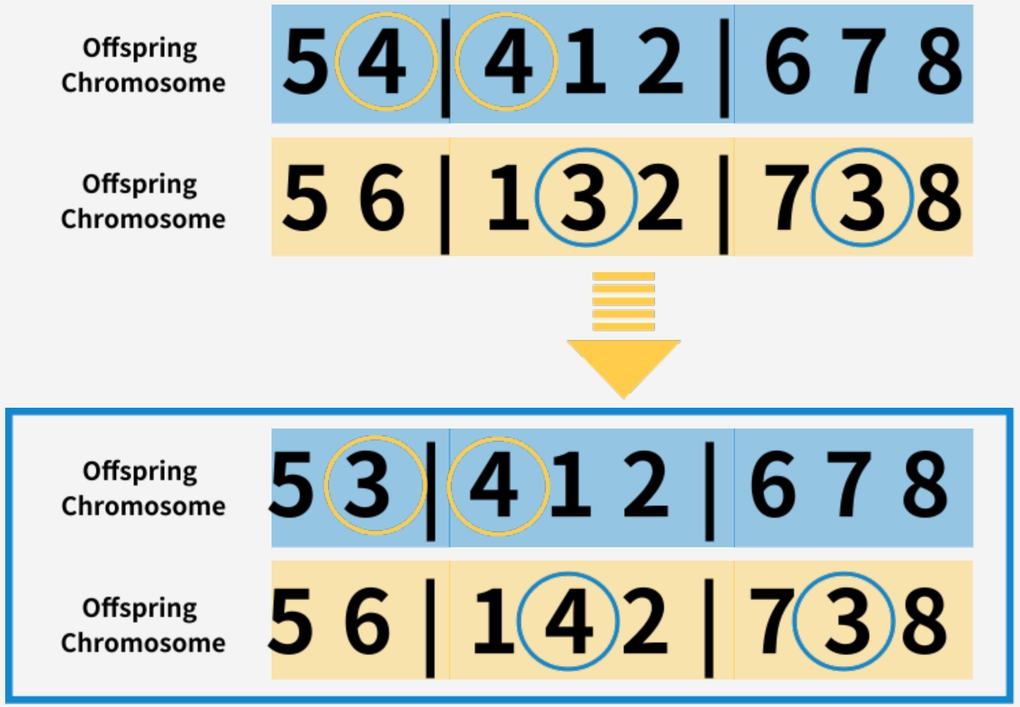
Selection

부모 population에서 **생존확률 p를 weight**으로
2개의 RANDOM Chromosome 선택
with replacement

Crossover

< PMX CROSSOVER >

STEP 2)



Mutation

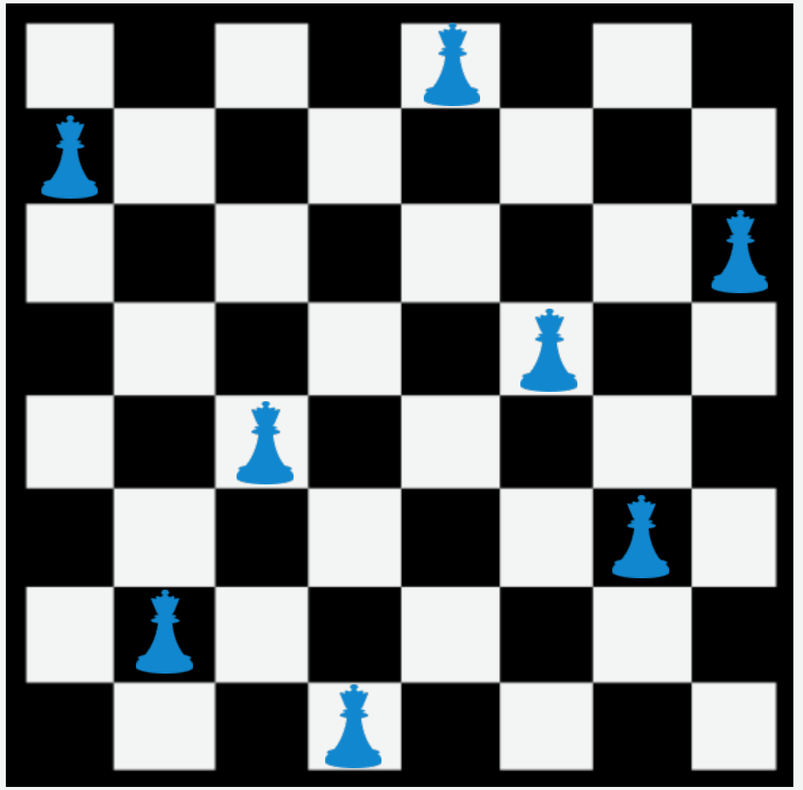
새로 생긴 자식 Chromosome의 두 부분을 랜덤으로 골라 위치를 서로 바꿔준다.

Replacement

새로운 (popSize)개의 population 생성
Fitness = 1 인 개체가 나올 때 까지
Selection, Crossover, Mutation 반복

Result | 실험 결과

1. 3-way tournament crossover



90 번째 8 queens Solution

Simulation으로 얻어진 8 - Queens Solution

	Solution	Iteration
1	17582463	1427
2	48136275	188
3	42857136	212
4	35841726	427
...	...	...
88	82417536	767
89	63175824	299
90	72418536	1
91	35281746	66
92	62714853	469

Result | 실험 결과

1. 3-way tournament crossover

논문의 simulation

n	max iteration
8	169369

우리의 simulation

n	max iteration	time(초)
5	256	0.038
6	409	0.335
7	19494	2.395
8	156172	21.618
9	2165311	306.223

2. OX crossover

```
library(GA)
ga(type = "permutation",
 crossover = "gaperm_oxCrossover",
seed = 2021, pmutation=0.1))
```

n	max iteration	time(초)
5	1	0.05
6	1	0.07
7	12	1.92
8	84	12.56
9	187	263.17

3. PMX crossover

```
library(GA)
ga(type = "permutation",
 crossover = "gaperm_pmxCrossover",
seed = 2021, pmutation=0.1))
```

n	max iteration	time(초)
5	1	0.027
6	1	0.03
7	17	1.779
8	67	7.755
9	192	299.899

Appendix | 구현 code

```
# 서로 공격하는 square수 구하기
attacking <- function(subject){
 error <- 0
 for (i in 1: length(subject)){
  x <- i
  y <- subject[i]
  for(j in 1:length(subject)){
   if (i !=j){
    dx <- j
    dy <- subject[j]
    if (abs(dx-x)==abs(dy-y)){error = error+1}
   }
  }
 }
 return(error)
}

# population 에서 임의로 3개 뽑아 fitness 구하기
# fitness 높은 순으로 2개 뽑기
selection ← function(population){
 candidate ← sample(population,3,replace = FALSE)
 fit ← list(candidate = NA, fitness = NA)
 for(i in 1:length(candidate)){
  fit$candidate[i] = list(candidate[[i]])
  fit$fitness[i] = 1/(attacking(candidate[[i]])+1)
 }
 top2 ← which(fit$fitness %in% sort(fit$fitness,decreasing=TRUE)[1:2])
 parents = c(candidate[top2[1]], candidate[top2[2]])
 return(c(parents, fit))
}

# 두 개의 chromosomes 일치성 확인
noncommon ← function(par1,par2){
 noncommon ← which(!par1-par2 == 0)
}
```

```
# Crossover operator : 3- way crossover
cross.operator<- function(par1,par2,nproblem){
 son <-c()
  # 두 par의 다른 부분 index
 offseed <- sample(par1[noncommon(par1,par2)])
 for (i in 1:nproblem){
  if (i %in% noncommon(par1,par2)) {
   son[i] <- offseed[1]
   offseed <- offseed[-1]}
  else {son[i] <- par1[i]}}
 return (son)
}
# Mutation operator
mutation.operator <- function(son,nproblem,pmut=0.1){
 randomdummy <- runif(1)
 if (randomdummy<=(pmut)){
  n.mut <- sample(1:nproblem,2)
  son.mut <- son
  son.mut[n.mut[1]] <- son[n.mut[2]]
  son.mut[n.mut[2]] <- son[n.mut[1]]
 }
 else {son.mut <- son}
 return(son.mut)
}
```

Appendix | 구현 code

```
# N-Queens Function
n_queens <- function(nproblem,n_sol){
  # Initial Population
  ipop <- n_sol
  son <-c()
  sol <- c()
  population1 <- c()
  for(i in 1:ipop){
    population1[[i]] <- sample(seq(1:nproblem), replace = FALSE)}
  solution <- data.frame(solution = rep(0,n_sol) , iter = rep(0,n_sol),
total_iter = rep(0,n_sol))
  i <- 0
  total_iter <- 0
  repeat{
    population <- population1
    iter <- 0
  repeat{
    population <- population1
    iter <- 0
  repeat{
    population <- population1
    iter <- 0
    # Creating further population ------------------------------
    repeat{
      result<- selection(population) # 한번 selection 돌려 결과 저장
      par1 <- result[[1]]
      par2 <- result[[2]]
      if (max(result$fitness) == 1) break
      #simulation 중 부모가 같은게 되는 경우 par1 랜덤하게 숫자배열 바꾸기
      com1 <- as.numeric(paste(par1,collapse=""))
      com2 <- as.numeric(paste(par2,collapse=""))
      if(com1 == com2){par1 <- sample(seq(1:nproblem), replace = FALSE)}
      crossed <- cross.operator(par1,par2,nproblem)
      son <- mutation.operator(crossed,nproblem) #문제가 일어나는 곳
      population <- c()
      population <- list(par1,par2,son)
      #if (iter > 100000) break
      iter <- iter+1
      result
    }

    sol_idx <- which(result$fitness == 1)
    sol = result$candidate[[sol_idx]]
    sol <- as.numeric(paste(sol,collapse=""))
    # print(iter)
    total_iter <- total_iter + iter
    if(sol %in% solution$solution){}else{ i <- i+1 ;solution$solution[i] <- sol;solution$iter[i] <- iter; solution$total_iter[i] <- total_iter}
    if (i == n_sol) break
  }
  return(solution)
}
```

```
# 5x5 Queens to 9x9 Queens

sol5 <- n_queens(5,10)
q5 <- system.time(n_queens(5,10))

sol6 <- n_queens(6,4)
q6 <- system.time(n_queens(6,4))

sol7 <- n_queens(7,40)
q7 <- system.time(n_queens(7,40))

sol8 <- n_queens(8,92)
q8 <- system.time(n_queens(8,92))

sol9 <- n_queens(9,352)
q9 <- system.time(n_queens(9,352))}
```

Appendix | GA Package 이용한 code

```
# 패키지를 이용한 GA
subject <- c(1,4,3,5,2,6,7,0)
attacking <- function(subject){
  error <- 0
  for (i in 1: length(subject)){
    x <- i
    y <- subject[i]
    for(j in 1:length(subject)){
      if (i !=j){
        dx <- j
        dy <- subject[j]
        if (abs(dx-x)==abs(dy-y)){error = error+1}
      }
    }
  }
  return(error)
}
fitness <- function(subject){
  fit = 1/(attacking(subject)+1)
  return(fit)
}
gaperm_Population(GA)
```

```
# default ox crossover
system.time(GA0 <- ga(type = "permutation", fitness = fitness,
maxiter = 72,seed = 2021, lower = 1, upper = 8 , popSize = 200
0, keepBest = TRUE, pmutation = 1))
summary(GA0)
View(GA0@bestSol)

# pmx crossover
system.time(GA <- ga(type = "permutation", fitness = fitness, c
rossover = "gaperm_pmxCrossover", maxiter = 69,
                     seed = 2021, lower = 1, upper = 8 , popSize = 2000,
keepBest = TRUE, pmutation = 1))
summary(GA)
View(GA@bestSol)
```

Thank You