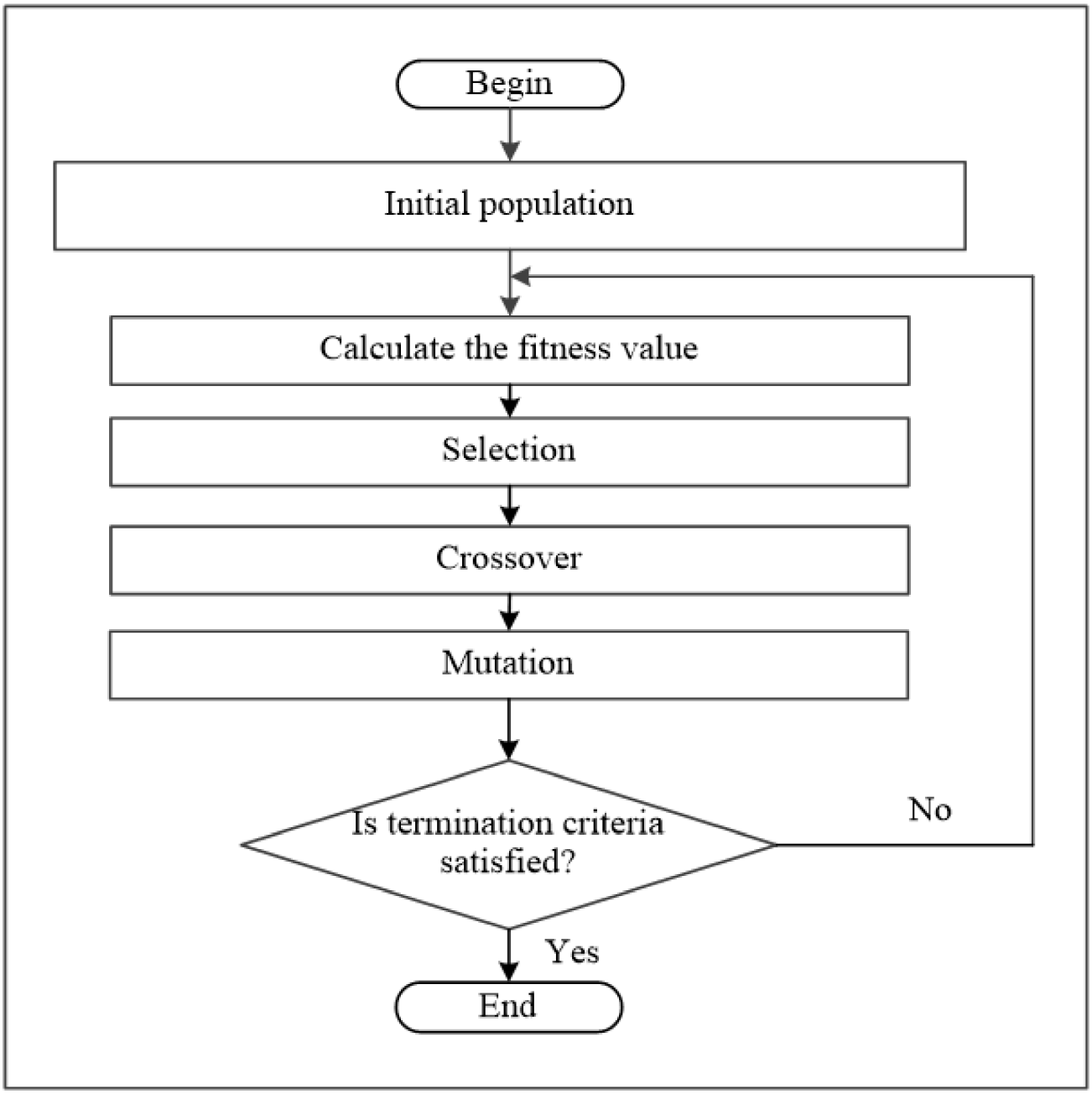

Create Initial Sudoku Board

1. create 2d empty array with numpy -> array of zeros **zero means cell is empty**
2. fill the created board with random values to start
3. save locations that have starting values to make them never change

Functions

1. create Board Function

```
def createInitialBoard():
    # some logic here
    return board
```

```
theBoard = createInitialBoard()
```

Create Initial Population *generation zero*

Now we have ready board to work with. let's say our population is 100 solutions.

1. create random solutions to be the first generation

Functions

1. Create Population

```
def createPopulation():
    # some logic here
    return population

initialPopulation = createPopulation()
```

Evaluation *applying fitness function on each individual in the population*

1. apply fitness function on each individual in the population **Fitness Function**
2. sort individuals depending on results of fitness function

Functions

1. Fitness Function

```
def getFitness(population):
    for individual in population:
        rowsFitness = checkRowsFitness(individual)
        columnsFitness = checkColumnsFitness(individual)
        blocksFitness = checkBlocksFitness(individual)
        fitness = overallFitness(rowsFitness, columnsFitness, blocksFitness)
    # some logic here
    return fitness

populationFitness = getFitness(population)
```

---

2. check rows fitness function

```
def checkRowsFitness(individual):
    # some logic here
    return rowsFitness
```

---

3. check columns fitness function

```
def checkColumnsFitness(individual):
    # some logic here
    return columnsFitness
```

---

4. check blocks Fitness

```
def checkBlocksFitness():
    # some logic here
    return blocksFitness
```

---

5. overall Fitness

```
def overallFitness(rowsFitness, columnsFitness, blocksFitness):
    # some equation uses all parameters
    return overallFitness
```

Select the Best depending on Evaluation

1. select the best individuals from population to continue living **Selection Function**

Functions

1. Selection Function

```
# must be sorted population got from fitness function
def selectParents(population):
    # logic here
    return parents
```

Cross Over to get Children

1. cross over two parents and create new children **cross over function**

Functions

1. crossover Function

```
# father and mother are individuals from selected population using selection
function
def crossOver(father, mother):
    # logic here
    return child
```

Update Children --> *mutation*

1. mutate each individual to be better

Functions

1. Mutation Function

```
# the individual passed to it must be one of children created by crossover
function
def mutate(individual):
    # logic here
    return betterIndividual
```