ANALISIS PERSOALAN KLASIK “THE N-QUEENS PROBLEM” MENGGUNAKAN ALGORITMA GENETIKA DAN ALGORITMA BACKTRACKING

Laporan

Ditujukan untuk memperoleh nilai pada

mata kuliah strategi algoritma

Oleh :

1214057 - Maylinda Christy Yosefina Talan

1214058 – Firda Yulianti

PROGRAM DIPLOMA IV TEKNIK INFORMATIKA

UNIVERSITAS LOGISTIK DAN BISNIS INTERNASIONAL

BANDUNG

2023

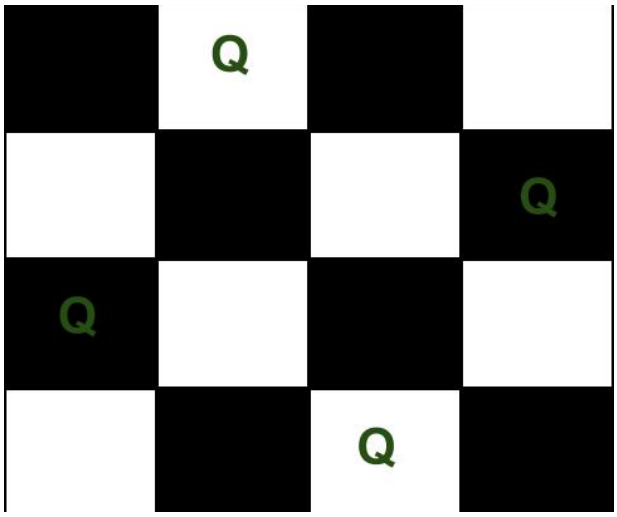
DAFTAR ISI

BAB I

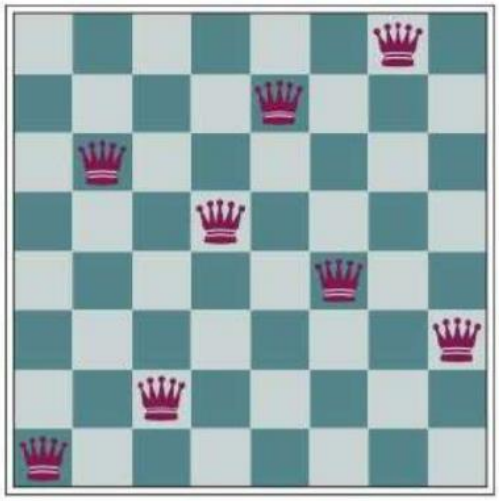
PENDAHULUAN

1.1 The N-Queens Problem

The N-Queens problem atau Persoalan N-Ratu adalah permasalahan di mana harus mencari cara bagaimana meletakkan Queen sebanyak n pada papan berukuran n x n sedemikian rupa sehingga tidak ada satu queen yang saling memakan dengan 1 langkah (Arrummaisha Adrifina). Sederhananya, N Queen adalah masalah menempatkan N ratu catur di papan catur N×N sehingga tidak ada dua ratu yang saling menyerang. Sebagai contoh, berikut adalah solusi untuk masalah 4 Ratu (Muhammad Khoirussolih, 2015).



Masalah 8 ratu itu sederhana. Pada papan catur 8x8, ratu dapat memindahkan sejumlah kotak secara horizontal, vertikal, dan diagonal. Biasanya, ada satu ratu per sisi di papan tetapi untuk masalah ini, ada 8. Ini dapat digeneralisasikan ke N ratu di papan NxN. Tujuannya adalah untuk menempatkan 8 ratu di papan sedemikian rupa sehingga setiap pasangan ratu tidak saling menyerang. Pada gambar di atas, Anda dapat melihat bahwa setiap ratu tidak berhadapan dengan ratu lainnya. Berikut adalah contoh status dewan yang bukan solusinya:



Pasangan ratu di kolom ke-4 dan ke-7 saling serang secara diagonal.

1.2 Algoritma Genetika

Algoritma genetika adalah sebuah metode optimisasi yang terinspirasi dari mekanisme seleksi alam pada organisme hidup. Algoritma ini digunakan untuk menyelesaikan masalah optimisasi yang sulit dan kompleks dengan cara mengevaluasi dan mengoptimalkan serangkaian solusi kandidat yang mungkin. Algoritma genetika menggunakan konsep-konsep dari teori evolusi untuk menghasilkan solusi yang semakin baik melalui beberapa generasi (Aditya Pratama Putra, 2021).

Algoritma genetika bekerja dengan cara membuat populasi awal dari individu-individu acak, yang biasanya dikenal sebagai kromosom. Setiap kromosom mewakili solusi potensial untuk masalah yang sedang dipecahkan. Selanjutnya, algoritma genetika menggunakan teknik-teknik seperti seleksi, rekombinasi, dan mutasi untuk menciptakan generasi berikutnya dari kromosom yang semakin baik.

Proses algoritma genetika terdiri dari beberapa tahap, yaitu:

1. Inisialisasi populasi awal dengan kromosom acak
2. Evaluasi kinerja kromosom dalam populasi
3. Seleksi kromosom terbaik untuk menghasilkan keturunan
4. Rekombinasi kromosom untuk menciptakan keturunan baru
5. Mutasi kromosom untuk menambah variasi dalam populasi
6. Evaluasi kinerja kromosom dalam populasi baru
7. Ulangi proses ini untuk beberapa generasi, hingga mencapai solusi yang optimal atau konvergensi.

Algoritma genetika dapat digunakan untuk menyelesaikan berbagai jenis masalah optimisasi, seperti masalah perencanaan produksi, penjadwalan tugas, optimasi portofolio investasi, dan banyak lagi salah satunya adalah permasalahan N-Queen (Cao Jianli, 2020).

1.3 Algoritma Backtracking

Algoritma backtracking adalah metode yang digunakan dalam pemrograman untuk menemukan solusi untuk masalah yang kompleks dengan cara mencoba semua kemungkinan solusi secara sistematis dan kemudian mengevaluasi kembali pilihan-pilihan tersebut untuk menemukan solusi yang optimal (Kiswanto, 2020).

Dalam algoritma backtracking, solusi ditemukan melalui pencarian yang sistematis, dengan menguji setiap kemungkinan secara berurutan dan memeriksa apakah setiap kemungkinan tersebut memenuhi persyaratan yang diberikan. Jika solusi yang diuji tidak memenuhi persyaratan, maka algoritma kembali ke tahap sebelumnya dan mencoba solusi alternatif. Proses ini terus diulang sampai solusi yang tepat ditemukan atau seluruh kemungkinan telah diuji.

Contoh masalah yang dapat dipecahkan dengan menggunakan algoritma backtracking antara lain:

1. Masalah Jalan Raya (Maze)
2. Masalah Penugasan (Assignment Problem)
3. Masalah Jalan Raya Terpendek (Shortest Path)
4. Masalah Knapsack (Membawa barang dengan bobot tertentu)
5. Masalah N-Queens (Menempatkan N ratu pada papan catur) (Purba, 2017)

BAB II

ANALISIS CODE

2.1 Analisis N-Queen dengan Algoritma Genetika

Pada penelitian ini, peneliti menggunakan code penyelesaian permasalahan pada N-Queen yang berasal dari github (sumber : https://github.com/mahdihassanzade/N-Queen-Problem-using-Genetic-Algorithm). Dengan code yang digunakan adalah sebagai berikut :

```
      from operator import indexOf
import random
```

Code diatas adalah library yang digunakan pada permasalahan N-Queen ini.

```
def random_chromosome(size):
    return [random.randint(0, size - 1) for _ in range(size)]
```

Code diatas digunakan untuk membuat kromosom pada permasalahan yang akan diselesaikan.

```
def fitness(chromosome, maxFitness):
    horizontal_collisions = (
        sum([chromosome.count(queen) - 1 for queen in chromosome]) / 2
    )
    diagonal_collisions = 0

    n = len(chromosome)
    left_diagonal = [0] * (2 * n - 1)
    right_diagonal = [0] * (2 * n - 1)
    for i in range(n):
        left_diagonal[i + chromosome[i] - 1] += 1
        right_diagonal[len(chromosome) - i + chromosome[i] - 2] += 1

    diagonal_collisions = 0
    for i in range(2 * n - 1):
        counter = 0
        if left_diagonal[i] > 1:
            counter += left_diagonal[i] - 1
        if right_diagonal[i] > 1:
            counter += right_diagonal[i] - 1
        diagonal_collisions += counter
```

```
    # 28-(2+3)=23
    return int(maxFitness - (horizontal_collisions + diagonal_collisions))
```

Code diatas digunakan untuk menghitung kalkulasi yang digunakan pada penyelesain permasalahan pada N-Queen.

```
def crossover(x, y):
    n = len(x)
    child = [0] * n
    for i in range(n):
        c = random.randint(0, 1)
        if c < 0.5:
            child[i] = x[i]
        else:
            child[i] = y[i]
    return child
```

Code diatas digunakan untuk melakukan crossover antara 2 kromosom yang sudah terbentuk.

```
def mutate(x):
    n = len(x)
    c = random.randint(0, n - 1)
    m = random.randint(0, n - 1)
    x[c] = m
    return x
```

Code diatas digunakan untuk membuat secara acak nilai indeks yang terdapat pada kromosom.

```
def probability(chromosome, maxFitness):
    return fitness(chromosome, maxFitness) / maxFitness
```

Code diatas digunakan untuk membuat menghitung probabilitas pada kromosom yang sudah dibuat.

```
def random_pick(population, probabilities):
    populationWithProbabilty = zip(population, probabilities)
    total = sum(w for c, w in populationWithProbabilty)
    r = random.uniform(0, total)
    upto = 0
    for c, w in zip(population, probabilities):
        if upto + w >= r:
```

```
            return c
        upto += w
    assert False, "Shouldn't get here"
```

Code diatas digunakan untuk membuat pemilihan roda roulette pada penyelesaian yang akan dibuat.

```
def genetic_queen(population, maxFitness):
    mutation_probability = 0.1
    new_population = []
    sorted_population = []
    probabilities = []
    for n in population:
        f = fitness(n, maxFitness)
        probabilities.append(f / maxFitness)
        sorted_population.append([f, n])

    sorted_population.sort(reverse=True)

    # Elitism
    new_population.append(sorted_population[0][1])  # the best gen
    new_population.append(sorted_population[-1][1])  # the worst gen

    for i in range(len(population) - 2):

        chromosome_1 = random_pick(population, probabilities)
        chromosome_2 = random_pick(population, probabilities)

        # Creating two new chromosomes from 2 chromosomes
        child = crossover(chromosome_1, chromosome_2)

        # Mutation
        if random.random() < mutation_probability:
            child = mutate(child)

        new_population.append(child)
        if fitness(child, maxFitness) == maxFitness:
            break
    return new_population
```

Code diatas digunakan untuk membuat algoritma genetic pada permasalahan yang akan diselesaikan.

```
def print_chromosome(chrom, maxFitness):
    print(
```

```
        "Chromosome = {},  Fitness = {}".format(str(chrom), fitness(chrom,
maxFitness))
    )
```

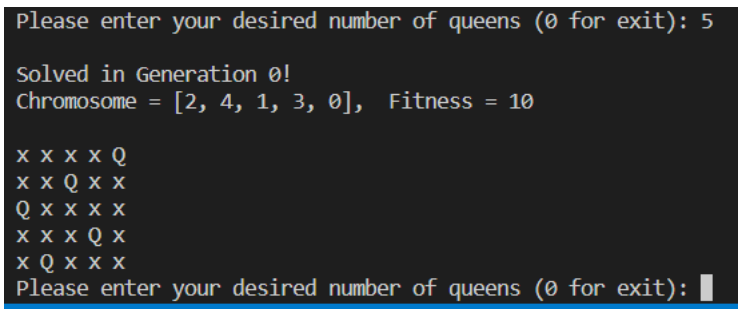
Code diatas digunakan untuk mencetak kromosom yang diberikan pada codingan sebelumnya.

```
def print_board(chrom):
    board = []

    for x in range(nq):
        board.append(["x"] * nq)

    for i in range(nq):
        board[chrom[i]][i] = "Q"

    def print_board(board):
        for row in board:
            print(" ".join(row))

    print()
    print_board(board)


if __name__ == "__main__":
    POPULATION_SIZE = 500

    while True:
        # say N = 8
        nq = int(input("Please enter your desired number of queens (0 for
exit): "))
        if nq == 0:
            break

        maxFitness = (nq * (nq - 1)) / 2  # 8*7/2 = 28
        population = [random_chromosome(nq) for _ in range(POPULATION_SIZE)]

        generation = 1
        while (
            not maxFitness in [fitness(chrom, maxFitness) for chrom in
population]
            and generation < 200
        ):

            population = genetic_queen(population, maxFitness)
            if generation % 10 == 0:
                print("=== Generation {} ===".format(generation))
```

```
                print(
                    "Maximum Fitness = {}".format(
                        max([fitness(n, maxFitness) for n in population])
                    )
                )
            generation += 1

        fitnessOfChromosomes = [fitness(chrom, maxFitness) for chrom in
population]

        bestChromosomes = population[
            indexOf(fitnessOfChromosomes, max(fitnessOfChromosomes))
        ]

        if maxFitness in fitnessOfChromosomes:
            print("\nSolved in Generation {}!".format(generation - 1))

            print_chromosome(bestChromosomes, maxFitness)

            print_board(bestChromosomes)

        else:
            print(
                "\nUnfortunately, we could't find the answer until generation
{}. The best answer that the algorithm found was:".format(
                    generation - 1
                )
            )
            print_board(bestChromosomes)
```

Code diatas digunakan untuk membuat cetakan diberi papan kromosom pada permasalahan sehingga outpur bisa keluar

Hasil dari code yang dibuat adalah sebagai berikut :



2.2 Analisis N-Queen dengan Algoritma Backtracking

Pada penelitian ini, peneliti menggunakan code penyelesaian permasalahan pada N-Queen yang berasal dari github (sumber : https://github.com/waqqasiq/n-queen-problem-using-backtracking). Dengan code yang digunakan adalah sebagai berikut :

```
import copy
import random
```

Code diatas adalah library yang digunakan pada permasalahan N-Queen ini.

```
def take_input():
    #Accepts the size of the chess board
    while True:
        try:
            n = int(input('Input size of chessboard? n = '))
            if n <= 3:
                print("Enter a value greater than or equal to 4")
                continue
            return n
        except ValueError:
            print("Invalid value entered. Enter again")
```

Code diatas digunakan untuk membuat board catur untuk menginput jumlah N yang akan kita masukkan dalam permasalahan tersebut.

```
def get_board(n):
    #Returns an n by n board
    board = ["x"]*n
    for i in range(n):
        board[i] = ["x"]*n
    return board
```

Code diatas digunakan untuk memberikan nilai n yang sudah diinput pada papan catur sebelumnya.

```
def print_solution(solutions, n):
    #Prints one of the solutions randomly
    x = random.randint(0,len(solutions)-1) #0 and len(solutions)-1 are
inclusive
    for row in solutions[x]:
        print(" ".join(row))
```

Code diatas digunakan untuk mencetak salah satu solusi secara acak pada N yang sudah diinput.

```
def solve(board, col, n):
    #Use backtracking to find all solutions
    if col >= n:
        return

    for i in range(n):
        if is_safe(board, i, col, n):
            board[i][col] = "Q"
            if col == n-1:
                add_solution(board)
                board[i][col] = "x"
                return
            solve(board, col+1, n) #recursive call
            #backtrack
            board[i][col] = "x"
```

Code diatas digunakan untuk membuat algoritma backtracking untuk menemukan semua solusi yang memungkinkan.

```
def is_safe(board, row, col, n):
    #Check if it's safe to place a queen at board[x][y]
    #check row on left side
    for j in range(col):
        if board[row][j] == "Q":
            return False

    i, j = row, col
    while i >= 0 and j >= 0:
        if board[i][j] == "Q":
            return False
        i=i-1
        j=j-1

    x, y = row,col
    while x < n and y >= 0:
        if board[x][y] == "Q":
            return False
        x=x+1
        y=y-1

    return True
```

Code diatas digunakan untuk memeriksa apakah aman untuk menempatkan queen pada papan X atau Y.

```
def add_solution(board):
    #Saves the board state to the global variable: solutions
    global solutions
    saved_board = copy.deepcopy(board)
    solutions.append(saved_board)
```

Code diatas digunakan untuk menyimpan status papan ke variabel global yang artinya untuk membuat suatu solusi.

```
n = take_input()
```

Code diatas digunakan untuk mengambil ukuran papan catur dari pengguna atau pemain permasalahan N-Queen.

```
board = get_board(n)
```

Code diatas digunakan untuk membuat atau mengembalikan papan persegi berdimensi nxn.

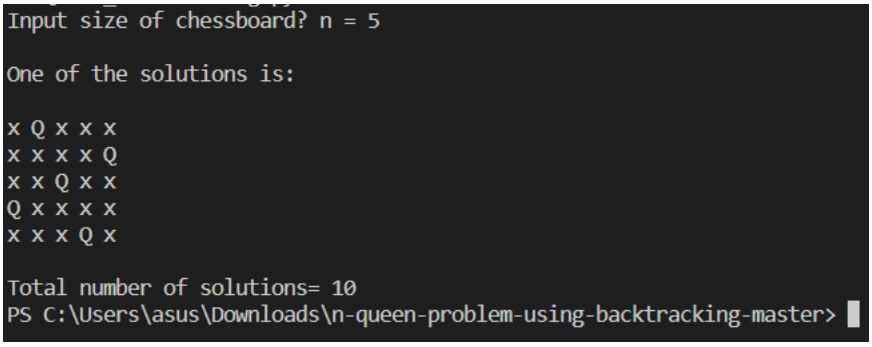
```
solutions = []
```

Code diatas digunakan untuk daftar kosong dari semua solusi yang memungkinkan dari N yang sudah diinput.

```
solve(board, 0, n)

print()

print("One of the solutions is: \n")
print_solution(solutions, n)

print()
print("Total number of solutions=", len(solutions))
```

Code diatas digunakan untuk menyelesaikan permasalah N-Queen dnegan algoritma backtracking dan menampilkan solusi dari permasalahan tersebut.

Hasil dari code yang dibuat adalah sebagai berikut :


Input size of chessboard? n = 5
One of the solutions is:
x Q x x x
x x x x Q
x x Q x x
Q x x x x
x x x Q x
Total number of solutions= 10
PS C:\Users\asus\Downloads\n-queen-problem-using-backtracking-master>

BAB III

PERBANDINGAN

3.1 Maylinda Christy Yosefina Talan

Masalah N-Queen adalah masalah penempatan N buah ratu pada papan catur berukuran NxN sedemikian sehingga tidak ada dua ratu yang saling menyerang dalam satu baris, satu kolom, dan diagonalnya. Algoritma genetik dan algoritma backtracking dapat digunakan untuk menyelesaikan masalah ini dengan pendekatan yang berbeda.

Algoritma genetik dapat digunakan dengan cara mengkodekan solusi sebagai kromosom dan memperbaiki populasi kromosom dengan metode seleksi, rekombinasi, dan mutasi. Setiap kromosom merepresentasikan solusi unik untuk masalah N-Queen, dan setiap generasi baru mencoba untuk memperbaiki populasi dengan mencari kromosom yang memiliki fitness tertinggi. Namun, karena kompleksitas masalah N-Queen meningkat secara eksponensial dengan ukuran papan catur, maka penggunaan algoritma genetik memerlukan waktu yang lama dan penggunaan memori yang signifikan untuk mencari solusi optimal.

Sementara itu, algoritma backtracking digunakan untuk menyelesaikan masalah N-Queen dengan mencoba setiap kemungkinan solusi secara berurutan dan menghindari pilihan yang tidak valid. Algoritma backtracking mampu menyelesaikan masalah N-Queen dengan cara yang lebih cepat dan lebih efisien dibandingkan algoritma genetik, terutama untuk ukuran papan catur yang lebih kecil. Namun, algoritma backtracking masih memerlukan waktu yang lama untuk menyelesaikan masalah pada ukuran papan catur yang lebih besar dan akan mengalami masalah ketika solusi optimal tidak ditemukan.

Dalam kesimpulannya, penggunaan algoritma genetik dan algoritma backtracking pada masalah N-Queen memiliki kelebihan dan kekurangan masing-masing. Algoritma genetik dapat digunakan untuk menemukan solusi optimal dalam waktu yang lebih lama, tetapi memerlukan penggunaan memori yang signifikan. Algoritma backtracking, di sisi lain, dapat digunakan untuk menyelesaikan masalah dengan cara yang lebih cepat dan lebih efisien, tetapi memiliki keterbatasan dalam menyelesaikan masalah pada ukuran papan catur yang lebih besar.

3.2 Firda Yulianti

Permasalahan N-Queen adalah masalah di mana kita harus menempatkan N buah ratu pada sebuah papan catur ukuran N x N sehingga tidak saling menyerang. Kedua algoritma, genetic dan backtracking, dapat digunakan untuk menyelesaikan masalah ini. Perbedaan utama antara algoritma genetic dan algoritma backtracking dalam menyelesaikan permasalahan N-Queen adalah cara mereka mencari solusi.

Algoritma backtracking secara sistematis mencoba setiap kemungkinan solusi secara rekursif, mulai dari kolom pertama dan kemudian mencoba menempatkan ratu satu per satu ke kolom berikutnya. Ketika ia menemukan bahwa tidak ada kemungkinan solusi pada kolom tertentu, ia akan kembali ke kolom sebelumnya dan mencoba kemungkinan solusi lain pada kolom tersebut. Algoritma backtracking memerlukan waktu yang relatif lama terutama ketika nilai N sangat besar.

Algoritma genetic menggunakan konsep evolusi biologis untuk menemukan solusi. Pada awalnya, beberapa kromosom acak dibuat yang merepresentasikan kemungkinan solusi. Kemudian, beberapa generasi dibuat dengan mengaplikasikan operator-genetik seperti seleksi, crossover dan mutasi pada kromosom-kromosom ini. Setiap generasi akan menciptakan solusi yang lebih baik dengan menggabungkan kromosom-kromosom yang baik dari generasi sebelumnya. Dengan demikian, algoritma genetic dapat menemukan solusi yang lebih cepat, tetapi bisa jadi tidak selalu memberikan solusi optimal.

Secara umum, algoritma genetic lebih cepat dalam menyelesaikan masalah N-Queen dibandingkan algoritma backtracking. Namun, algoritma genetic memerlukan jumlah parameter yang lebih besar dan hasilnya mungkin tidak selalu optimal. Di sisi lain, algoritma backtracking sangat cocok untuk masalah N-Queen dengan ukuran kecil, tetapi mungkin memerlukan waktu yang lama untuk mencari solusi pada kasus yang lebih besar.

DAFTAR PUSTAKA

Aditya Pratama Putra, S. Y. (2021). Sistem Informasi Penentuan Rute Pengiriman Barang di CV ASA Menggunakan Metode Algoritma Genetika. *KLIK: Kajian Ilmiah Informatika dan Komputer*, 35-42.

Arrummaisha Adrifina, P. S. (n.d.). PENYELESAIAN MASALAH N-QUEEN DENGAN TEKNIK BACKTRACKING.

Cao Jianli, C. Z. (2020). Parallel genetic algorithm for N-Queens problem based on message passing interface-compute unified device architecture. *International Journal Computational Intelligence*.

Kiswanto, R. H. (2020). Spesifikasi Komputer Rakitan Berdasarkan Kebutuhan dan Anggaran Menggunakan Algoritma Backtracking. *Jurnal Eksplora Informatika*.

Muhammad Khoirussolih, G. W. (2015). Penyelesaian Masalah 8-Queen Dengan Depth First Search Menggunakan Algoritma Backtracking.

Purba, F. (2017). PENERAPAN ALGORITMA RANUT BALIK (BACKTRACKING) DALAM N-QUEEN PROBLEM PERMAINAN CATUR.