

LOCAL SEARCH

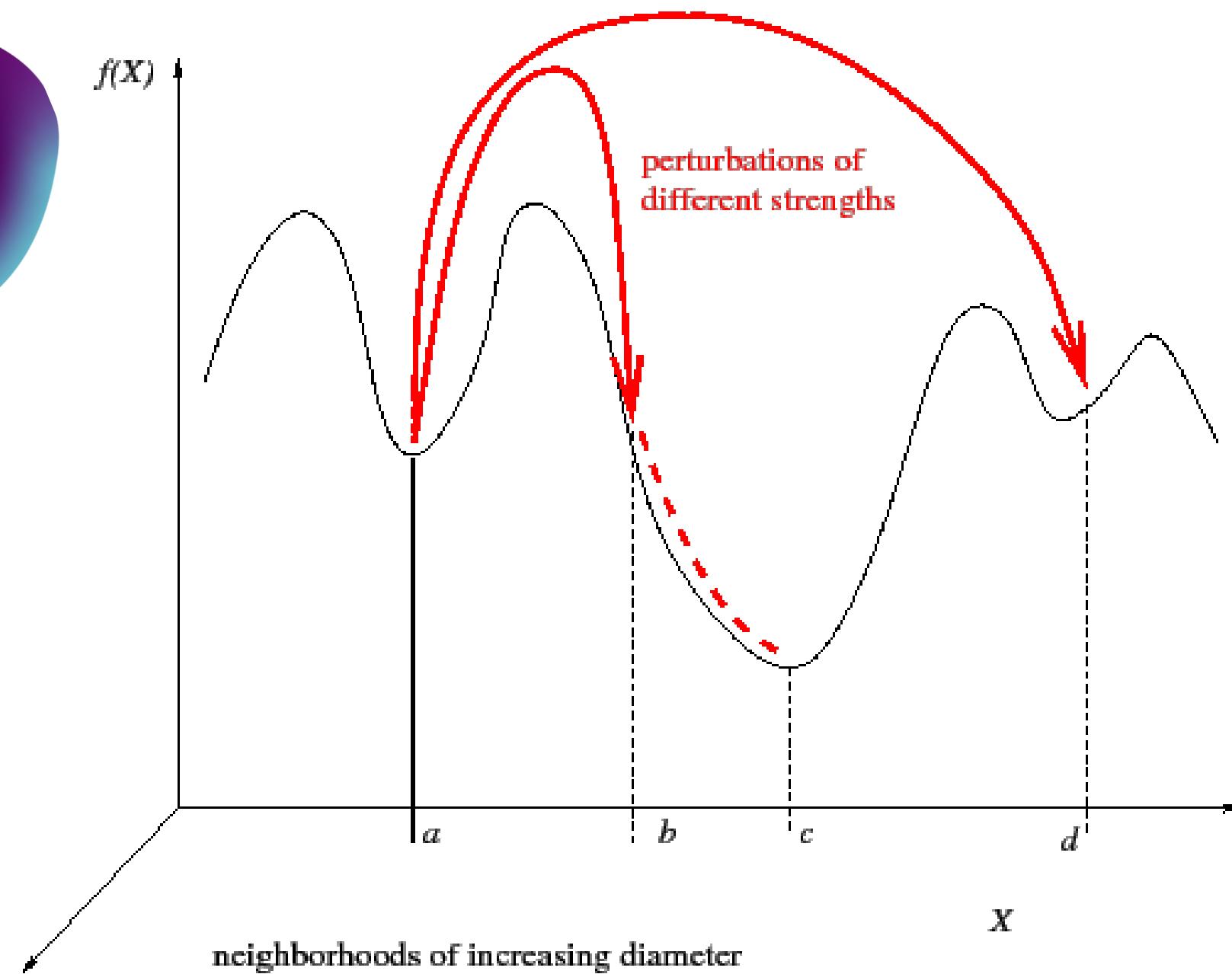
8-QUEENS

Timothy Hosia (5025211098)

Yohanes Teguh (5025211179)

Shazia Naveeda (5025211203)

LOCAL SEARCH IN 8-QUEENS



Local search adalah teknik pemecahan masalah optimisasi untuk menemukan solusi yang meminimalkan atau memaksimalkan suatu fungsi objektif dengan mempertimbangkan solusi sekitarnya secara iteratif.

Dalam konteks pemecahan masalah 8 queens, local search dapat digunakan untuk menemukan penempatan optimal dari 8 ratu pada papan catur 8x8, di mana tidak ada ratu yang saling menyerang.

Dalam local search untuk 8 queens, solusi diwakili oleh suatu vektor dengan 8 elemen, di mana setiap elemen merepresentasikan baris di mana seorang ratu ditempatkan pada kolom tertentu. Setiap iterasi local search, solusi baru dibuat dengan mengubah posisi satu atau lebih ratu pada papan catur dan mengevaluasi fungsi objektif (jumlah pasangan ratu yang saling menyerang). Jika solusi baru lebih baik dari solusi sebelumnya, solusi baru tersebut akan diterima sebagai solusi sementara. Proses ini diulang sampai tidak ada perubahan yang dapat dilakukan lagi pada solusi. Local search merupakan teknik heuristik, yang artinya solusi yang ditemukan tidak selalu merupakan solusi optimal yang sebenarnya. Namun, local search cukup efektif dalam menyelesaikan masalah 8 queens karena kompleksitas perhitungan yang lebih rendah daripada teknik pemecahan masalah lainnya seperti backtracking.

IMPLEMENTASI

```
1 import random
2
3
4 def calculate_conflicts(board):
5     """
6         Fungsi untuk menghitung jumlah konflik pada board yang diberikan
7     """
8     conflicts = 0
9     n = len(board)
10    for i in range(n):
11        for j in range(i+1, n):
12            if board[i] == board[j]:
13                conflicts += 1
14            offset = j - i
15            if board[i] == board[j] - offset or board[i] == board[j] + offset:
16                conflicts += 1
17    return conflicts
```

Fungsi `calculate_conflicts(board)` digunakan untuk menghitung jumlah konflik pada suatu board. Konflik terjadi ketika dua atau lebih queen berada pada baris, kolom, atau diagonal yang sama.

```
19  def get_random_board(n):
20      """
21          Fungsi untuk menghasilkan sebuah board acak dengan n queen
22      """
23      board = list(range(n))
24      random.shuffle(board)
25      return board
```

Fungsi `get_random_board(n)` digunakan untuk menghasilkan board acak dengan n queen. Board awal yang dihasilkan secara acak akan digunakan sebagai kondisi awal untuk pencarian solusi dengan local search.

```
27 def get_neighbour_boards(board):
28     """
29     Fungsi untuk menghasilkan semua kemungkinan tetangga dari board yang diberikan
30     """
31     neighbour_boards = []
32     n = len(board)
33     for i in range(n):
34         for j in range(i+1, n):
35             neighbour = board.copy()
36             neighbour[i], neighbour[j] = neighbour[j], neighbour[i]
37             neighbour_boards.append(neighbour)
38     return neighbour_boards
```

Fungsi `get_neighbour_boards(board)` digunakan untuk menghasilkan semua kemungkinan tetangga dari suatu board. Tetangga diperoleh dengan menukar posisi dua queen pada board. Fungsi ini akan mengembalikan sebuah list of list, yang masing-masing elemen listnya merepresentasikan satu tetangga dari board yang diberikan.

```
40 def local_search(n, max_iterations):
41     """
42         Fungsi utama untuk menyelesaikan 8 Queen dengan local search
43     """
44     current_board = get_random_board(n)
45     current_conflicts = calculate_conflicts(current_board)
46     for i in range(max_iterations):
47         if current_conflicts == 0:
48             # Jika tidak ada konflik, maka solusi sudah ditemukan
49             return current_board
50     neighbour_boards = get_neighbour_boards(current_board)
51     best_neighbour = None
52     best_neighbour_conflicts = n*n
53     for neighbour in neighbour_boards:
54         neighbour_conflicts = calculate_conflicts(neighbour)
55         if neighbour_conflicts < best_neighbour_conflicts:
56             best_neighbour = neighbour
57             best_neighbour_conflicts = neighbour_conflicts
58         if best_neighbour_conflicts >= current_conflicts:
59             # Jika tidak ada tetangga yang lebih baik, maka berhenti
60             return current_board
61     current_board = best_neighbour
62     current_conflicts = best_neighbour_conflicts
63     # Jika sudah mencapai maksimum iterasi, kembalikan board terbaik yang ditemukan
64     return current_board
```

Fungsi `local_search(n, max_iterations)` merupakan fungsi utama yang akan mencari solusi dengan menggunakan algoritma local search. Fungsi ini menerima dua parameter yaitu `n` yang merepresentasikan jumlah queen, dan `max_iterations` yang merepresentasikan jumlah iterasi maksimum yang dilakukan dalam pencarian solusi.

Pencarian solusi dilakukan dengan cara menghasilkan sebuah board acak, kemudian mencari tetangga-tetangganya satu per satu untuk mencari tetangga terbaik yang memiliki jumlah konflik paling sedikit. Jika tetangga terbaik memiliki jumlah konflik yang lebih sedikit dibandingkan dengan board saat ini, maka akan digunakan tetangga terbaik sebagai board saat ini dan proses akan berlanjut. Namun jika tidak ada tetangga yang lebih baik, maka pencarian akan berhenti dan board saat ini akan dikembalikan sebagai solusi.

```
66 # Fungsi untuk mencetak board dalam bentuk matriks
67 def print_board(board):
68     n = len(board)
69     for i in range(n):
70         row = ["."]*n
71         row[board[i]] = "Q"
72         print(" ".join(row))
```

Fungsi `print_board(board)` digunakan untuk mencetak board dalam bentuk matriks dengan queen direpresentasikan oleh simbol "Q" dan cell kosong direpresentasikan oleh simbol "..".

```
73  
74 # Contoh penggunaan  
75 solution = local_search(8, 1000)  
76 if solution:  
77     print_board(solution)  
78 else:  
79     print("Solusi tidak ditemukan dalam 1000 iterasi")  
80
```

Di dalam contoh penggunaan, algoritma local search dipanggil untuk menyelesaikan masalah 8 Queen dengan melakukan pencarian solusi dalam 1000 iterasi. Jika solusi ditemukan, maka board solusi tersebut akan dicetak menggunakan fungsi `print_board(board)`. Jika solusi tidak ditemukan dalam 1000 iterasi, maka pesan "Solusi tidak ditemukan dalam 1000 iterasi" akan dicetak.



**TERIMA
KASIH**