



KECERDASAN BUATAN (F)

8-QUEEN LOCAL SEARCH (HILL CLIMBING) ALGORITHM

Rayhan Arvianta Bayuputra 5025211217
Arif Nugraha Santosa 5025211048
Laurivasya Gadhing Syahafidh 5025211136

Overview

OMSA



Hill Climbing Algorithm

Algoritma pencarian heuristik yang digunakan dalam kecerdasan buatan dan optimisasi untuk menemukan solusi optimal dalam ruang pencarian tertentu.

Kelebihan

Simplicity

Efficiency

Convergence

Hill Climbing Algorithm

Kelemahan

Random Restart

Problem Reformulation

Inconsistent Space

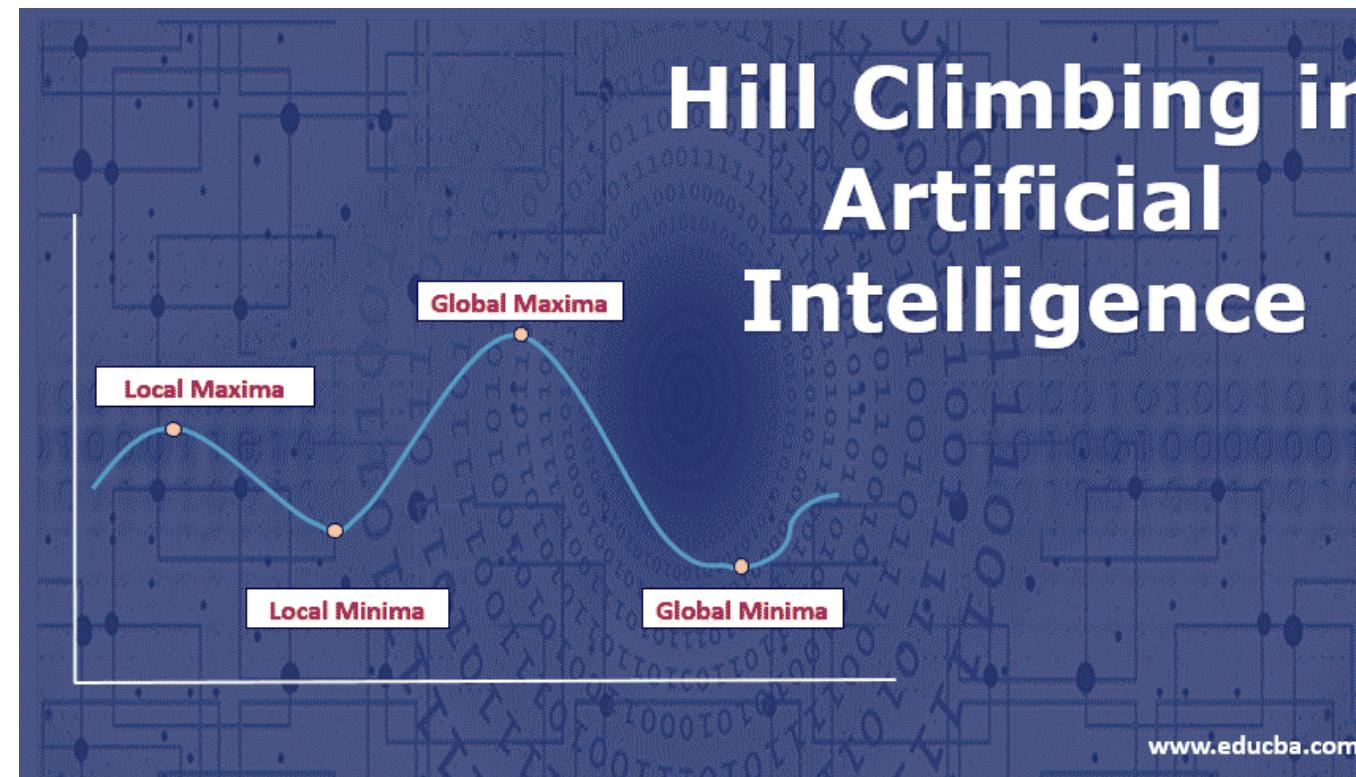
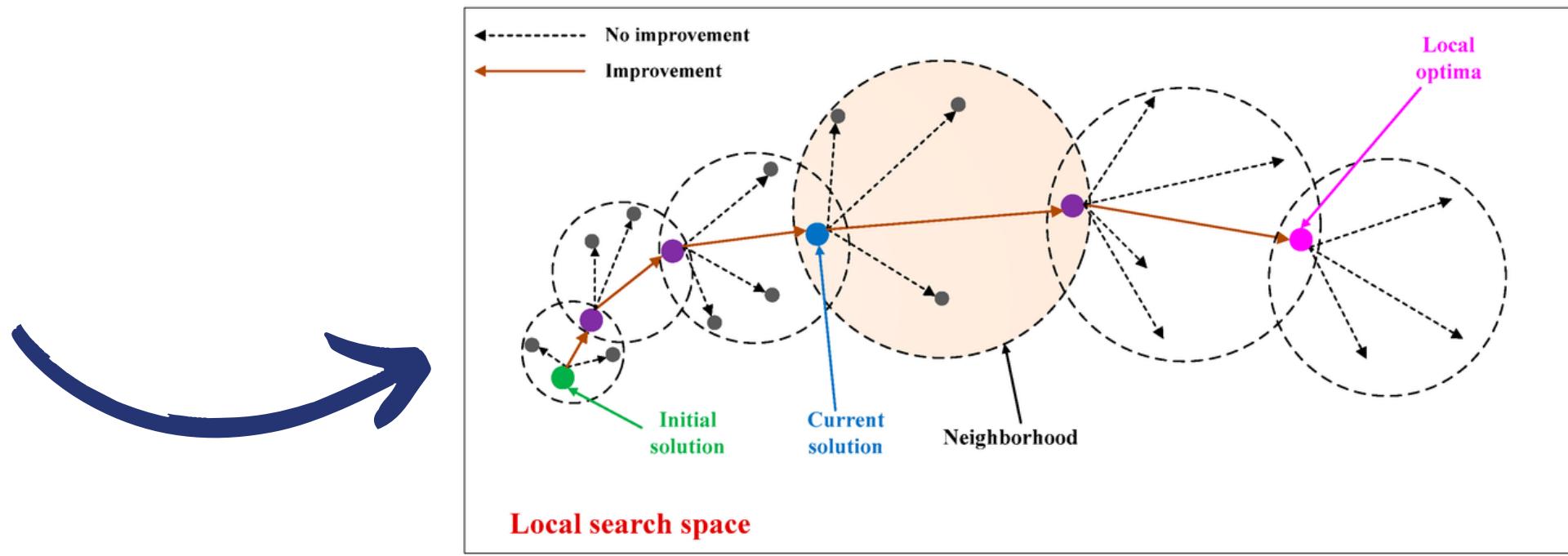
Implementasi

Menggunakan Hill Climbing Algorithm untuk mendapatkan solusi dari 8-queen

Local Search, Algoritma Hill Climbing

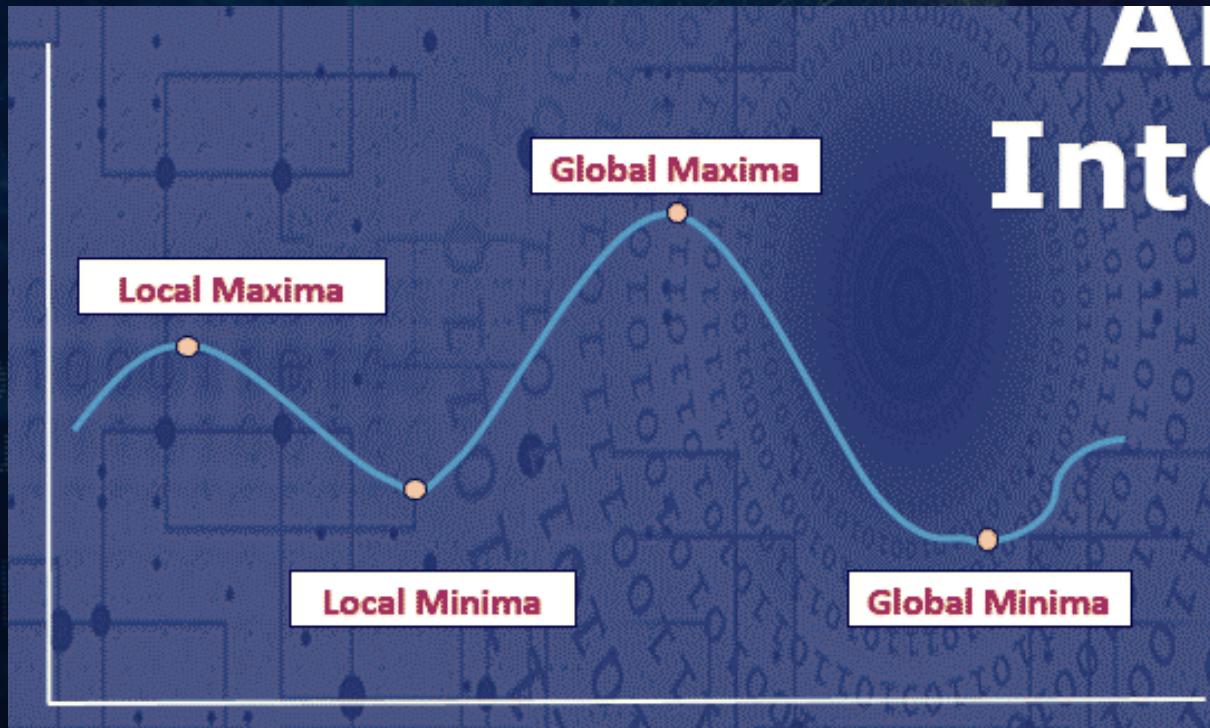


Local search atau pencarian lokal adalah metode algoritma pencarian solusi yang memfokuskan pada eksplorasi solusi yang ada di sekitar solusi saat ini, tanpa mempertimbangkan seluruh ruang solusi yang mungkin.



Hill climbing atau algoritma naik bukit adalah salah satu jenis algoritma pencarian lokal yang mencari solusi dengan mengambil langkah ke arah yang lebih baik secara bertahap, seperti mendaki bukit.

Hill Climbing Algorithm



KELEBIHAN

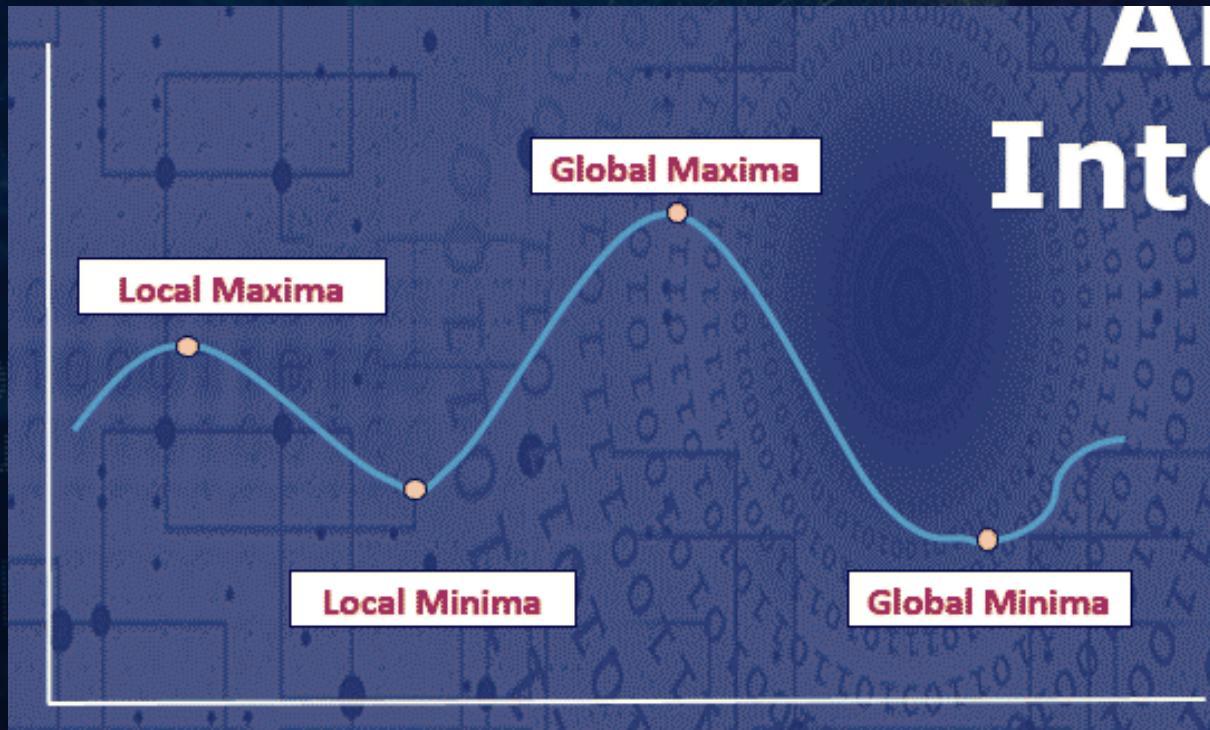
SIMPLICITY

Hill climbing adalah algoritma yang sederhana dan mudah diimplementasikan, sehingga menjadi pilihan yang menarik untuk menyelesaikan banyak masalah optimisasi.

EFFICIENCY

Hill climbing dapat sangat efisien untuk jenis masalah tertentu, terutama yang memiliki jumlah variabel yang kecil dan ruang pencarian yang terbatas.

Hill Climbing Algorithm



KELEBIHAN

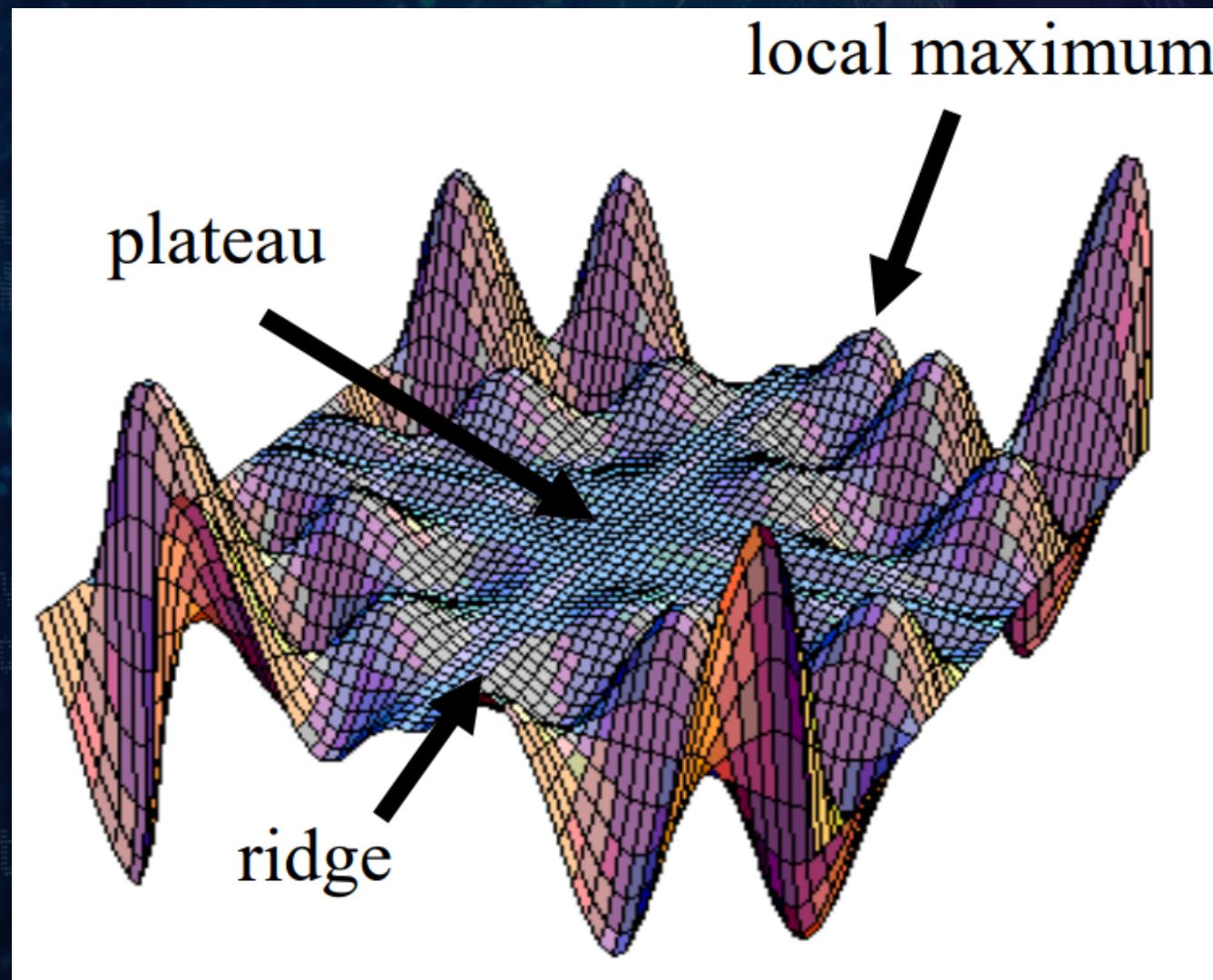
Convergence

Hill climbing akan selalu konvergen ke optimum lokal, yang berarti akan menemukan solusi terbaik di sekitar titik awal pencarian.

Flexibility

Hill climbing dapat dimodifikasi dan dikombinasikan dengan algoritma lain untuk meningkatkan kinerjanya dan mengatasi beberapa keterbatasannya.

Hill Climbing Algorithm



KEKURANGAN

Random Restart

Terus memulai kembali pencarian dari lokasi acak hingga tujuan ditemukan

Problem Reformulation

Merumuskan kembali search space untuk mengeliminasi problem tersebut

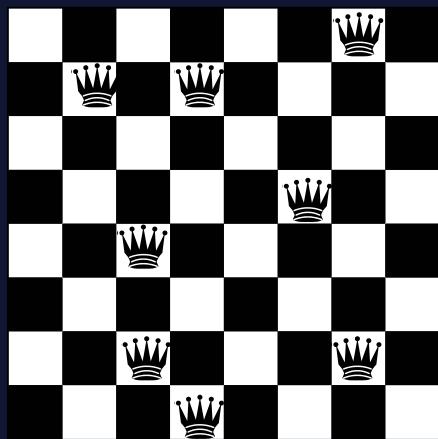
Inconsistent Space

Beberapa problem space bagus untuk hill climbing namun lainnya ada yang sangat buruk

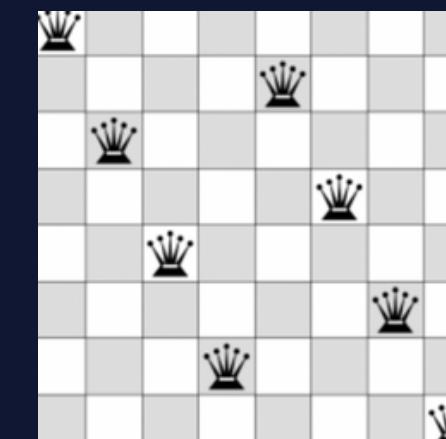
8-QUEEN

Terdapat sebuah papan catur yang masih kosong, problem yang akan kita selesaikan adalah meletakkan pion queen sebanyak 8 buah dengan posisi tertentu agar pion queen tidak bisa saling menyerang satu sama lain.

Ketentuan dalam permainan catur adalah queen bisa bergerak secara diagonal atau tegak lurus



STATE AWAL



STATE AKHIR

```
3 def get_attacking_pairs(board):  
4     # Count the number of attacking pairs of queens on the board  
5     n = len(board)  
6     count = 0  
7     for i in range(n):  
8         for j in range(i+1, n):  
9             if board[i] == board[j] or abs(board[i]-board[j]) == abs(i-j):  
10                 count += 1  
11     return count
```

Fungsi `get_attacking_pairs(board)` akan menghitung jumlah pasangan queen yang saling menyerang pada papan catur dengan ukuran 8x8. Fungsi ini akan menerima satu parameter yaitu sebuah list yang merepresentasikan posisi dari queen-queen pada papan catur. Untuk menghitung jumlah pasangan queen yang saling menyerang, fungsi ini menggunakan perulangan nested for yang akan membandingkan setiap queen satu persatu. Setiap pasangan queen yang diperiksa, akan diperiksa apakah queen tersebut saling menyerang atau tidak dengan menggunakan kondisi if. Jika queen-queen tersebut saling menyerang, maka jumlah pasangan queen yang saling menyerang akan bertambah satu. Setelah semua pasangan queen diperiksa, fungsi akan mengembalikan jumlah pasangan queen yang saling menyerang tersebut.

```
13 def print_solution(board):  
14     # Print the board as a chessboard with queens represented by 'Q'  
15     n = len(board)  
16     for i in range(n):  
17         for j in range(n):  
18             if board[i] == j:  
19                 print('Q', end=' ')  
20             else:  
21                 print('.', end=' ')  
22     print()
```

Fungsi `print_solution(board)` akan mencetak papan catur dengan queen-queen yang direpresentasikan dengan huruf Q, dan kotak kosong dengan titik(.) berdasarkan posisi dari queen-queen yang terdapat pada list yang diberikan sebagai parameter.

Fungsi ini menggunakan dua perulangan `for` yang akan mengakses setiap baris dan kolom pada papan catur. Untuk setiap posisi pada papan catur, fungsi akan mengecek apakah terdapat queen pada posisi tersebut atau tidak. Jika queen terdapat pada posisi tersebut, maka fungsi akan mencetak huruf Q, sedangkan jika tidak terdapat queen, maka fungsi akan mencetak titik.

Setelah selesai mencetak papan catur, fungsi akan selesai dan tidak mengembalikan nilai apapun.

```

24 def hill_climbing(initial_state):
25     board = initial_state
26     score = get_attacking_pairs(board)
27     print(f"Heuristic value for initial state: {score}")
28
29     while True:
30         neighbors = []
31         for i in range(8):
32             for j in range(8):
33                 if board[i] != j:
34                     neighbor = list(board)
35                     neighbor[i] = j
36                     neighbors.append(neighbor)
37
38         best_neighbor = board
39         best_score = score
40         for neighbor in neighbors:
41             neighbor_score = get_attacking_pairs(neighbor)
42             if neighbor_score < best_score:
43                 best_neighbor = neighbor
44                 best_score = neighbor_score
45
46         if best_score >= score:
47             print(f"Final heuristic value: {score}")
48             return board
49
50         board = best_neighbor
51         score = best_score
52         print(f"Heuristic value: {score}")

```

hill_climbing(initial_state): fungsi ini akan menerima sebuah list yang merepresentasikan posisi dari queen-queen di atas papan catur sebagai initial_state. Kemudian, fungsi ini akan mencoba menemukan solusi dengan menggunakan algoritma hill climbing. Algoritma ini akan mencari solusi dengan mencoba untuk bergerak dari state ke state yang lebih baik secara heuristik. Fungsi ini akan mengembalikan state yang ditemukan sebagai solusi.

1. Pertama-tama, fungsi akan menginisialisasi variabel board dengan initial_state. Kemudian, fungsi akan menghitung score dari initial_state menggunakan fungsi get_attacking_pairs(board) dan mencetak score tersebut.
2. Selama algoritma belum menemukan solusi, fungsi akan mencari semua state yang mungkin di sekitar state saat ini. State ini akan disimpan dalam list neighbors.

```

24 def hill_climbing(initial_state):
25     board = initial_state
26     score = get_attacking_pairs(board)
27     print(f"Heuristic value for initial state: {score}")
28
29     while True:
30         neighbors = []
31         for i in range(8):
32             for j in range(8):
33                 if board[i] != j:
34                     neighbor = list(board)
35                     neighbor[i] = j
36                     neighbors.append(neighbor)
37
38         best_neighbor = board
39         best_score = score
40         for neighbor in neighbors:
41             neighbor_score = get_attacking_pairs(neighbor)
42             if neighbor_score < best_score:
43                 best_neighbor = neighbor
44                 best_score = neighbor_score
45
46         if best_score >= score:
47             print(f"Final heuristic value: {score}")
48             return board
49
50         board = best_neighbor
51         score = best_score
52         print(f"Heuristic value: {score}")

```

4. Kemudian, fungsi akan mencari state yang paling baik secara heuristik dari semua neighbors. State yang paling baik ini akan disimpan dalam variabel best_neighbor dan score heuristiknya akan disimpan dalam variabel best_score.
5. Jika score heuristik dari best_neighbor lebih buruk atau sama dengan score heuristik dari state saat ini, maka fungsi akan menghentikan pencarian dan mengembalikan state saat ini sebagai solusi.
6. Jika score heuristik dari best_neighbor lebih baik daripada score heuristik dari state saat ini, maka fungsi akan mengubah variabel board menjadi best_neighbor dan mengubah variabel score menjadi best_score. Kemudian, fungsi akan mencetak score heuristik dari best_neighbor.

```
54 # Generate a random initial state and run the hill climbing algorithm from it
55 initial_state = [random.randint(0, 7) for _ in range(8)]
56 print(f"Initial State : {initial_state}")
57 print_solution(initial_state)
58 print("\n")
59
60 solution = hill_climbing(initial_state)
61
62 print(f"\nSolution : {solution}")
63 print_solution(solution)
```

program akan menghasilkan solusi dengan memanggil fungsi `hill_climbing(initial_state)` menggunakan `initial_state` yang di-generate secara acak. Solusi ini kemudian akan dicetak menggunakan fungsi `print_solution(board)`.

OUTPUT 8-QUEENS (HILL CLIMBING)

```
Initial State : [3, 6, 5, 2, 6, 4, 0, 1]
. . . Q . . .
. . . . . Q .
. . . . Q . .
. . Q . . . .
. . . . . Q .
. . . . Q . .
Q . . . . . .
. Q . . . . .

Heuristic value for initial state: 5
Heuristic value: 4
Heuristic value: 3
Final heuristic value: 3

Solution : [3, 1, 5, 2, 6, 4, 7, 1]
. . . Q . . .
. Q . . . .
. . . . Q . .
. . Q . . . .
. . . . . Q .
. . . . Q . .
. . . . . Q
. Q . . . .
```