

Analisa Penyelesaian Permainan 8-Queen dengan Algoritma *Backtracking* dan Algoritma *Hill Climbing*

Endah Sulisthyani

Fakultas Sains dan Teknologi
Universitas Al Azhar Indonesia
Kebayoran Baru, Jakarta Selatan
endahsulisthyani@gmail.com

Abstrak—*Artificial Intelligent* dapat membantu suatu program untuk bekerja dengan pemikiran program itu sendiri, sehingga meminimalisir campur tangan penggunaannya. Penggunaan AI ini terdapat pada permainan catur. 8-Queen merupakan merupakan salah satu bentuk permainan *puzzle* dengan persoalan menempatkan 8 ratu pada papan catur berukuran 8 x 8. Makalah ini membahas tentang bagaimana perbedaan antara penyelesaian 8-Queen dengan menggunakan *backtracking* dengan *hill climbing*.

Kata Kunci— *Artificial Intelligent; Queen; Backtracking; Hill Climbing;*

I. PENDAHULUAN

Kecerdasan Buatan atau Intelegensi Artifisial didefinisikan sebagai kecerdasan entitas yang ilmiah. Sistem seperti ini umumnya dianggap seperti komputer. Kecerdasan diciptakan dan dimasukkan ke dalam suatu mesin atau computer agar dapat melakukan pekerjaan seperti yang dilakukan manusia. Banyak algoritma digunakan untuk membuat *Artificial Intelligence* pada permainan papan seperti catur, *checker* dan sebagainya.

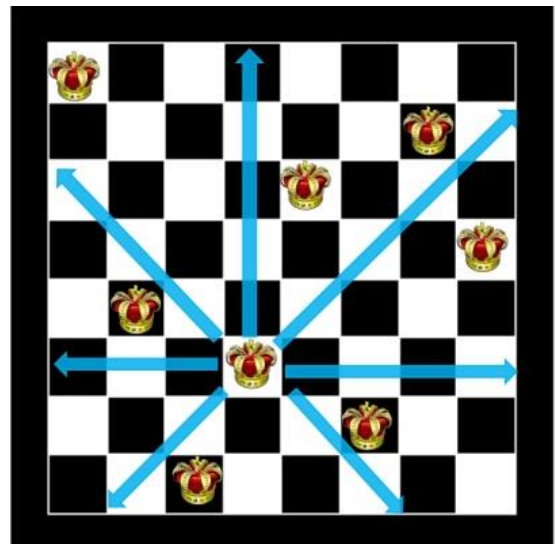
Permainan catur merupakan permaianan yang mengasah otak. Bentuknya yang sederhana tidak mengurangi kompleksitas dalam mengatur strategi dan cara untuk memainkannya. Salah satu usaha yang dilakukan untuk memenangkan permainan ini yaitu dengan meminimalkan kekalahan dengan mempunyai pengetahuan mengenai letak atau posisi dari bidak-bidak catur yang aman dan tidak terancam. Salah satu problem yang ada dan menarik untuk diteliti yaitu masalah *n-queen* problem (8-queen).

8-Queen merupakan merupakan salah satu bentuk permainan *puzzle* dengan persoalan menempatkan 8 ratu pada papan catur berukuran 8 x 8 sedemikian rupa sehingga tidak ada ratu yang menyerang ratu lainnya. Seorang ratu dapat bergerak secara horizontal (di baris yang sama), vertikal (di kolom yang sama) dan diagonal sama, seperti diilustrasikan pada Gambar 1. Permasalahan yang terjadi pada 8-Queen harus mengikuti peraturan berikut:

1. Ada paling banyak satu ratu di setiap kolom.
2. Ada paling banyak satu ratu di setiap baris.
3. Ada paling banyak satu ratu di setiap diagonal.

Permainan *puzzle* 8-Queen, pertama kali dibentuk pada tahun 1848 oleh seorang pemain catur Jerman, Max Bezzel. Dari tahun ke tahun, banyak matematikawan termasuk C.F

Gauss dan George Cantor telah bekerja keras untuk dapat menyelesaikan masalah *N-Queen* ini. Pada tahun 1850, Solusi pertama kali dibentuk oleh Franz Nauck, dengan menemukan semua 92 solusi. Nauck juga memperluas *puzzle* ke bentuk *N-Queen* [1].



Gambar 1. Pergerakan arah ratu

Dalam proses penyelesaiannya permainan ini banyak terdapat algoritma-algoritma pencarian yang dapat diterapkan. Pada makalah ini akan membahas tentang penggunaan algoritma *backtracking* dan algoritma *hill climbing* dalam menemukan langkah terbaik ratu untuk tidak ada ratu yang mengancam satu sama lain dan apa perbedaan antara menggunakan metode *backtracking* dan *hill climbing*.

B. Rumusan Masalah

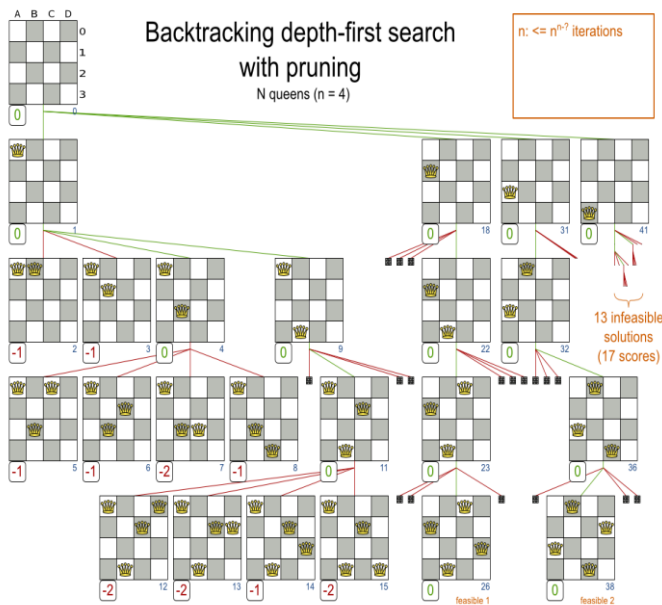
Pada tugas akhir makalah dengan judul “Analisa Penyelesaian Permainan 8-Queen dengan Algoritma *Backtracking* dan Algoritma *Hill Climbing*” memiliki rumusan masalah sebagai berikut :

1. Apakah penggunaan *Backtracking* dan *Hill Climbing* dapat menemukan langkah terbaik ratu untuk tidak ada ratu yang mengancam satu sama lain?
2. Bagaimana perbedaan antara menggunakan metode *Backtracking* dan *Hill Climbing* ?

II. LANDASAN TEORI

A. Algoritma *Backtracking*

Backtracking adalah teknik yang digunakan untuk memecahkan masalah dengan pencarian yang besar secara sistematis mencoba dan menghilangkan kemungkinan [7]. Jika bisa diterapkan, *backtracking* jauh lebih cepat daripada pencarian *brute force*, karena bisa menghilangkan sejumlah besar kandidat hanya dengan satu tes tunggal. *Backtracking* adalah alat yang sangat penting untuk memecahkan masalah seperti *Crosswords*, *Sudoku* dan berbagai jenis teka-teki dll. Seringkali proses yang sangat efektif untuk memecahkan masalah ransel dan beberapa masalah pengoptimalan lainnya. Itu tergantung pada prosedur yang ditetapkan pengguna, yang menentukan masalahnya; sifat kandidat parsial dan bagaimana mereka berakhir menjadi solusi lengkap. Oleh karena itu, bukan algoritma yang spesifik. Walaupun, tidak seperti banyak algoritma non-spesifik lainnya, algoritma ini dijamin untuk menemukan semua solusi pemecahan masalah yang mungkin terjadi dalam jumlah waktu yang relatif sedikit.



Gambar 2. Proses *Backtracking*

B. Algoritma *Hill Climbing Search*

Hill Climbing Search terkadang disebut *Greedy Local Search* karena mencari keadaan tetangga yang baik tanpa memikirkan ke depan tentang ke mana harus pergi selanjutnya. Meski keserakahan dianggap salah satu dari tujuh dosa mematikan, ternyata algoritma *greedy* itu sering tampil cukup

baik. *Hill climbing* sering membuat solusi karena biasanya cukup mudah memperbaiki keadaan buruk.

Algoritma *steepest-ascent hill climbing* ini merupakan satu-satunya komponen yang terus bergerak menuju peningkatan nilai yaitu menanjak. Ini berakhir ketika mencapai "puncak" di mana tidak ada tetangga yang memiliki nilai lebih tinggi. Algoritma tidak mempertahankan pohon pencarian, sehingga struktur data untuk node saat ini hanya perlu mencatat keadaan dan nilai dari fungsi objektif. Pendakian Hill tidak melihat ke depan melampaui tetangga langsung dari keadaan saat ini [2].



Gambar 3. *Local search: Hill Climbing*

III. ANALISA

A. Pencarian solusi dengan *Backtracking*

Berdasarkan konsep kombinatorial, untuk sembarang nilai N, *N-Queens Problem* memiliki kemungkinan penyelesaian seperti rumus yang dibawah ini [8]:

$$\left(\frac{(N * N)!}{(N * (N - 1))!} \right)$$

Apabila kemudian diterapkan syarat bahwa dua buah bidak ratu tidak dapat terletak dalam satu kolom yang sama, maka nilai tersebut dapat dikurangi menjadi N^N kemungkinan solusi.

Mekanisme algoritma *Backtracking* yaitu akan mencari semua solusi yang mungkin, sehingga pertama-tama harus dibuat algoritma dasar yang dapat melakukan terhadap semua kemungkinan solusi. Algoritma ini dibuat untuk menelusuri kemungkinan solusi pada suatu pohon solusi abstrak. Algoritma *Backtracking* dianggap sebagai perbaikan dari algoritma *Brute-Force* karena pada *Backtracking* penelusuran terhadap cabang-cabang dapat dihentikan jika pada suatu titik cabang tertentu diketahui bahwa penelusuran tersebut tidak akan mencapai solusi yang diinginkan. Dengan demikian, kompleksitas program dapat dikurangi.

Langkah menggunakan algoritma *backtracking* [3] :

1. Mulailah di kolom paling kiri
2. Jika semua ratu ditempatkan kembali benar
3. Cobalah semua baris di kolom saat ini. Ikuti setiap baris yang dicoba.
 - a) Jika ratu dapat ditempatkan dengan aman di baris ini maka tandai [baris, kolom] ini sebagai bagian dari solusi dan rekursif periksa apakah menempatkan ratu di sini mengarah ke solusi.
 - b) Jika menempatkan ratu di [baris, kolom] mengarah ke solusi maka kembalilah dengan benar.
 - c) Jika menempatkan ratu tidak mengarah pada solusi maka unmark ini [baris, kolom] (*Backtrack*) dan lanjutkan ke langkah (a) untuk mencoba baris lainnya.
4. Jika semua baris telah dicoba dan tidak ada yang berhasil, return false untuk memicu kemunduran.

Pseudocode:

```

Procedure n_queen(input N:integer)
{   Input:N = jumlah queen
  Output: semua solusi      x      =
(x[1], x[2], ..., x[N])
}
Deklarasi
    row : integer

Algoritma:
    row ← 1
    col[1] ← 0
    while row ≤ 0 do
        col[row] ← col[row]+1
        while ((col[row]) ≤ and (not
placed(row))) do
            col[row] := col[row]+1
        endwhile
        if col[row] ≤ N then
            if row = N then
                PrintSolution(row,N)
            else
                row ← row+1
                col[row] ← 0
            endif
        else
            row ← row-1
        endif
    endwhile
    
```

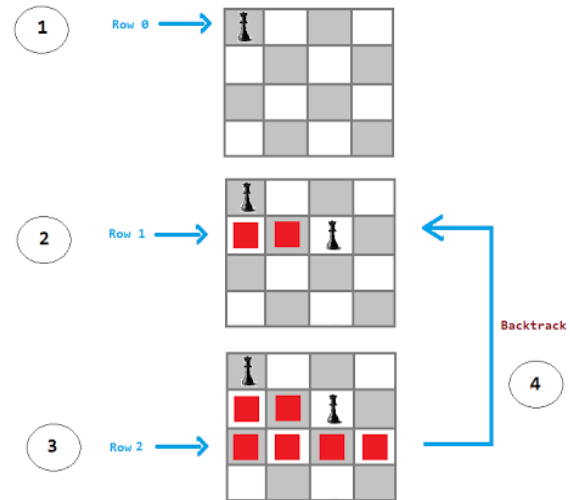
Untuk melihat lebih jauh aplikasi backtracking pada permasalahan ini, mari kita tinjau proses pencarian solusi untuk $N = 8$. Kemungkinan penempatan 8 buah bidak ratu pada suatu papan catur 8×8 secara sembarang tanpa ada syarat khusus adalah :

$$\left(\frac{(8 \cdot 8)!}{(8 \cdot (8-1))!} \right) = 1.784629876 \times 10^{14} \text{ kemungkinan.}$$

Jika kemudian diaplikasikan syarat bahwa tidak boleh ada dua dari delapan

bidak tersebut yang terletak dalam kolom yang sama, maka total solusi yang mungkin adalah $8^8 = 16777216$ kemungkinan, jika kita mencari solusi dengan manual.

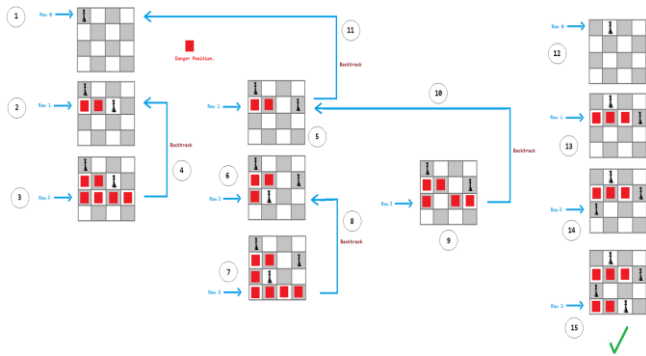
Untuk menggambarkan lebih jelas proses kerja algoritma backtracking, saya akan dimisalkan penempatan 4 bidak ratu pada papan catur berikut ini :



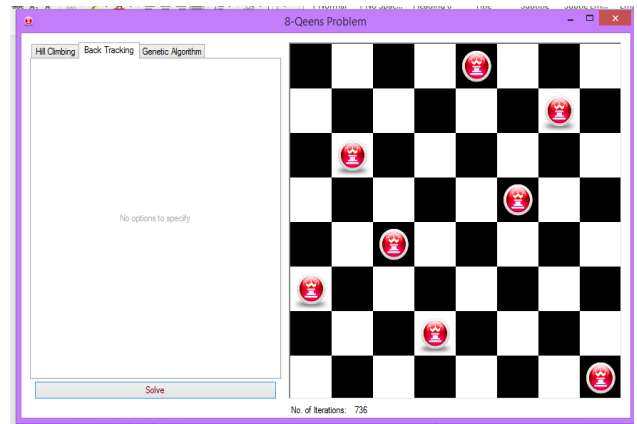
Gambar 4. Proses *Backtracking*

Berdasarkan proses algoritma yang telah disebutkan, maka dalam pencarian solusi, bidak pertama akan diletakkan pertama kali pada [baris 1, kolom 0]. Pencarian kemudian akan dilanjutkan ke [baris 1, kolom 1], dan pada [baris 1, kolom 1] dan [baris 2, kolom 1] akan otomatis dipangkas karena sudah tidak mungkin menghasilkan solusi karena [baris 1, kolom 1] terletak pada baris yang sama dengan bidak pertama, sedangkan [baris 2, kolom 1] terletak pada diagonal yang sama. Pada kolom 1, bidak akan diletakkan pada [baris 3, kolom 1], kemudian pencarian dilanjutkan ke [baris 1, kolom 2]. Dengan posisi bidak saat ini, tidak ada petak dalam [baris 1, kolom 2] yang bisa ditempati oleh bidak ke-tiga, sehingga program akan melakukan backtrack kembali ke kolom 1. [baris 3, kolom 1] akan dipangkas dan bidak pada kolom 1 akan dipindahkan ke petak [baris 4, kolom 1]. Dengan melanjutkan pencarian solusi menggunakan teknik serupa di atas, bidak pada kolom 2 akan diletakkan pada [baris 2, kolom 2].

Masalah akan kembali muncul ketika bidak akan ditempatkan di kolom 3, karena ternyata tidak ada petak yang dapat ditempati bidak ratu ke-empat. Akibatnya, program akan melakukan backtrack, memangkas [baris 2, kolom 3], [baris 3, kolom 3], dan [baris 4, kolom 3] karena dua yang terakhir juga tidak mengarah pada solusi, kemudian melakukan backtrack sampai kolom 0 karena pada kolom 1 seluruh petak telah dipangkas. Petak [baris 1, kolom 0] kemudian dipangkas dan pencarian berlanjut dengan penempatan bidak ratu pertama di petak [baris 2, kolom 0].



Gambar 4. Proses *Backtracking* Akhir



Gambar 5. Posisi Ratu Setelah Menemukan Solusi Terakhir

B. Pencarian solusi dengan *Hill Climbing*

Hill climbing bekerja dengan cara yang sama namun sama sekali mengabaikan memori simpul yang dieksplorasi. Oleh karena itu, ia menyusuri Search Tree dengan memilih penerus dengan nilai heuristik termurah, tanpa menyimpan memori negara yang dieksplorasi. Ini akan memastikan bahwa teknik heuristik berfungsi dengan penggunaan memori minimal, setidaknya perhitungan mungkin namun tetap mempertahankan keuntungan dari metode pencarian solusi yang diinformasikan.

Pseudocode :

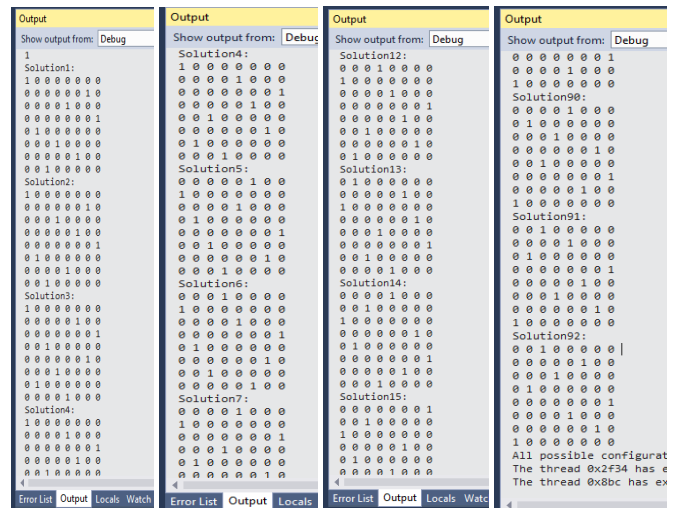
```
function HILL-CLIMBING(problem) returns a
solution state
  inputs: problem, a problem
  static: current, a node
         next, a node
  current ← MAKE-NODE (INITIAL-
STATE [problem])
  loop do
    next ← a highest-valued
successor of current
    if VALUE [next] <
VALUE [current] then return current
    current ← next
  end
```

Untuk kasus *8-Queen* ini, harus mencari tetangga terdekat dari ratu. Jika $h = 0$ tidak mendapatkan solusi. Namun ketika di $h=5$ baru mendapatkan solusi yang terbaik.

IV. HASIL DAN PEMBAHASAN

Hasil program aplikasi *8-Queen* ini :

Permasalahan pada *8-Queen* akan mendapatkan perhitungan solusi yang banyak kemungkinan yaitu $1.784629876 \times 10^{14}$ kemungkinan. Jika dengan menggunakan proses komputasi pencarian solusi *backtracking* hanya ada 92 solusi yang di dapatkan. *Source code* program *backtracking* ini di dapatkan dari [4]



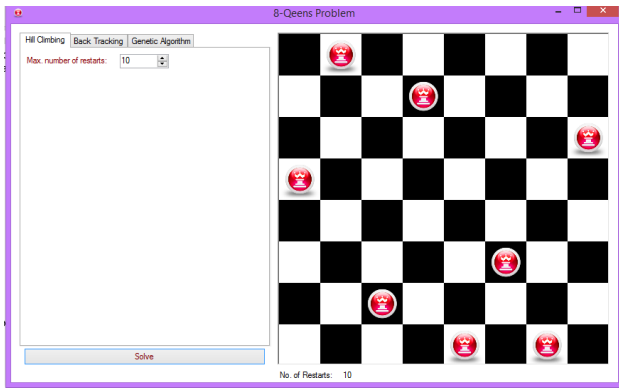
Gambar 6. Proses Iterasi Algoritma *Backtracking*

Untuk mengetahui solusi dari n yang berbeda kita bisa lihat Tabel 1. Program ini diambil dari program yang berbeda [5]

Tabel 1. Solusi *Backtracking*

n	Solusi	Time
8	92	1
9	352	1
10	724	1

Pada pencarian solusi *hill climbing* hasil yang di dapatkan adalah sebagai berikut:



Gambar 7. Proses *Hill Climbing*

Dengan proses iterasi dengan hasil:

Best board found this iteration:

```
00001000
00000001
01000000
00000010
00000000
00100000
10000000
00010100
```

pairs of queens attacking each other: 1

Dari data tersebut menemukan iterasi yang terbaik dengan pasangan ratu yang saling menyerang 1.

Untuk mengetahui hasil n yang berbeda terdapat di tabel 2..ini data diambil dari program berbeda[6]

Tabel 2. Hill Climbing

n	Iterasi	Langkah	Number of attacking	Time
8	8	3	5	12 detik
9	8	5	3	15 detik
10	8	5	5	9 detik

Dengan menggunakan teknik *backtracking* dapat dimungkinkan untuk melakukan jauh lebih baik karena Menjamin ditemukannya solusi (jika solusinya memang ada) dan solusi yang ditemukan pasti yang paling baik. Sedangkan menggunakan teknik *hill climbing* ini dengan pendekatan heuristik yang paling hemat memori namun memiliki risiko kegagalan yang tinggi karena masalah maxima lokal dan mengabaikan ingatan, terutama untuk solusi sedang sampai lama.

References

- [1] J. Bell and B. Stevens. A survey of known results and research areas for n-queens. *Discrete Mathematics*, 309(1):1–31, 2009.
- [2] S. Russell and P. Norvig. *Artificial Intelligence A Modern Approach* Third Edition. 2010.
- [3] Geeksforgeeks. Backtracking | set 3 (n queen problem). URL <https://www.geeksforgeeks.org/backtracking-set-3-n-queen-problem/>
- [4] R. Abdelrahman. I solved The 8 queens problem using Hill climbing & Backtracking & Genetic Algorithm using C#. URL: <https://drive.google.com/file/d/0B3i6QyCiM7YTZktDUjZxWkZOUgc/view>
- [5] S. S. Nazrul. N queen problem. URL https://github.com/snazrul1/PyRevolution/blob/master/Puzzles/N_Queen_Problem.ipynb.
- [6] Collinsmith. N Queen Solver. URL <https://github.com/collinsmith/n-Queen-Solver>
- [7] E. V. Khanna and E. S. Chopra. International journal of engineering sciences & research technology comparative analysis of backtracking, tuned hybrid technique and genetic algorithm for optimization of n-queens problem.
- [8] Stumpers. Treebeard's Stumper Answer 4 February 2000. URL <https://www.rain.org/~mkummel/stumpers/04feb00a.html>
- [9] B. Selman and C. P. Gomes. Hill-climbing search. *Encyclopedia of Cognitive Science*, 2006.
- [10] Madhusudan and R. Rangra. Probability-based solution to n-queen problem. *International Journal of Engineering Science and Innovative Technology (IJESIT)*, Volume 3, 2014.
- [11] B. S. Farhad Soleimanian and G. Feyzipour. A new solution for n queens problem using blind approaches: Dfs and bfs algorithms. *International Journal of Computer Applications (0975 - 8887)*, Volume 53, No.1, 2012.