Implementasi Algoritma Backtracking Pada Knight's Tour Problem

Akbar Serdano, Muhammad Zarlis dan Dedy Hartama
Teknik Informatika S2, Fakultas Ilmu Komputer dan Teknologi Informasi Universitas Sumatera Utara
Jl. Alumni No.9, Kampus USU Padang Bulan, Medan 20155
akbarserdano27@gmail.com

ABSTRAK

Algoritma Backtracking merupakan algoritma yang berhubungan pada DFS untuk menemukan solusi secara lebih mangkus dari pada algoritma brute force. Algoritma Backtracking secara umum menentukan solusi dengan cara memangkas (pruning) bagian yang tidak mengarah ke solusi. Dalam kehidupan sehari-hari, algoritma Backtracking sering diterapkan untuk menemukan lokasi yang dituju maupun masalah lainnya. Dalam penelitian ini algoritma Backtracking di implementasikan ke dalam suatu masalah mengenai Knight's Tour, dimana untuk menemukan solusi yang lebih baik. Knight's Tour adalah sebuah jalur yang akan di lewati oleh knight pada sebuah papan catur 8 x 8, dimana posisi dimulai dari titik yang ditentukan untuk melewati semua kotak tepat hanya satu kali dilalui pada papan catur 8 x 8. Hasil penelitian ini menunjukkan bahwa algoritma Backtracking dapat menemukan solusi yang lebih optimal dibandingkan algoritma brute force.

Kata Kunci : *Knight's Tour Problem, Algoritma Backtracking, Brute Force, Hamilton Path, Depth First Search.*

PENDAHULUAN

Dalam permainan catur terdapat dua orang pemain dalam suatu papan catur dimana terdiri dari 8 kolom dan 8 baris yang membentuk kotak berwarna hitam dan putih secara silang. Permainan catur mempunyai 1 raja, 1 ratu, 2 benteng, 2 peluncur, 2 kuda dan 8 bidak pada setiap pemain. Kemampuan dalam bermain catur mampu mengasah strategi dan logika berpikir saat menyerang lawan.

Mulai dari permainan catur ini banyak orang menciptakan permainan baru, yang dimana mengkombinasikan setiap komponen didalam permainan catur. Salah satunya permainan baru yang diciptakan adalah *Knight's Tour*. Dalam permainan *Knight's Tour* hanya menggunakan 1 kuda catur dan aturan permainan ini dimana 1 kuda harus melewati semua kotak tepat satu kali dengan berjalan membentuk huruf "L"[1].

Proses penyelesaian pada permainan *Knight's Tour* dapat dilakukan membentuk suatu grap, yang mana setiap kotak atau jalur yang dilalui dapat dikatakan edge. Node dan edge yang saling terhubung akan membentuk sebuah lintasan yang akan dilalui. Dalam permasalahan *Knight's Tour* dimana harus melewati semua titik tepat 1 kali tetapi tidak kembali ke titik semula atau disebut dengan jalur Hamilton (*Hamilton Path)*[1].

Pada penelitian mengenai permasalahn *Knight's Tour* telah banyak yang melakukan penelitian tersebut baik dari dalam negeri maupun luar negeri. Beberapa metode yang telah digunakan dalam menyelesaikan permasalahn *Knight's Tour* ini, salah satunya adalah Sulistyono yang berjudul "Penggunaan Teori Graf dan Algoritma *Backtrack* dalam Penyelesaian Masalah *Knight's Tour*" [2].

Algoritma *Backtracking* merupakan algoritma yang berhubungan pada DFS untuk menemukan solusi secara lebih mangkus dari pada algoritma *brute force*. Algoritma *Backtracking* secara umum menentukan solusi dengan cara memangkas (*pruning*) bagian yang tidak mengarah ke solusi dari semua kemungkinan pada solusi yang ada. Tujuan dari penelitian ini adalah mengimplementasikan algoritma *Backtracking* untuk menemukan solusi dari semua kemungkinan dari *Knight's Tour Problem* pada papan catur *m x n*. Pada penelitian ini menggunakan papan catur 8 x 8.

LANDASAN TEORI

Algoritma *Backtracking*

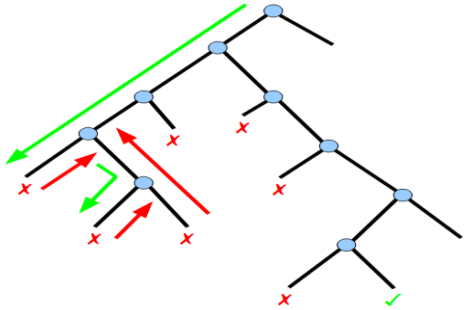
Istilah Algoritma *Backtracking* pertama kali diperkenalkan oleh D.H. Lehmer pada tahun 1950. Selanjutnya R.J. Walker, Golomb,dan Baumert melakukan kajian persoalan mengenai tentang *Backtracking* dan penerapannya. *Backtracking* merupakan perbaikan dari algoritma *brute force*. *Backtracking* menelusuri pohon solusi dengan prinsip DFS (*Depth First Search*).

Dalam pengambilan keputusan penelusuran, *Backtracking* hanya mempertimbangkan pilihan-pilihan yang mengarah kepada ditemukannya solusi. Hal ini merupakan keunggulan algoritma *Backtracking* atas algoritma *brute force*. Ciri khas dari algoritma *Backtracking* adalah adanya penelusuran kembali ke simpul sebelumnya apabila jalur yang dilalu mengalami kebuntuhan, lalu melanjutkan kemungkinan solusi lainnya.

Berikut ini uraian langkah yang dilakukan dalam pencarian solusi dengan algoritma *Backtracking* [3]:

- Solusi yang akan dicari dapat membentuk sebuah lintasan seperti tree ataupun graph. Aturan yang digunakan mengikuti aturan *depht-first order* (DFS).
- Simpul yang sudah dilahirkan yang memiliki *node* dinamakan **simpul hidup** (*live node*).
- Simpul yang sedang diperluas dinamakan **simpul-E** (*Expand node*).
- Tiap simpul-E yang akan diperluas, maka akan membangun sebuah lintasan terhadap *node* yang ada.
- Pada lintasan yang sedang diperluas tidak mengarahkan pada solusi maka simpul tersebut akan dijadikan **simpul mati** (*dead node*).
- Simpul-E yang lintasannya telah mati maka dapat menerapkan **fungsi pembatas**.
- Simpul yang telah mati tidak akan diperluas lagi.
- Jika lintasaan berakhir pada simpul mati makan proses backtrack dilakukan dengan melanjutkan ke simpul diatasnya.
- Lalu, meneruskan dengan memperluas simpul *node* lainnya.
- Selanjutnya simpul berikutnya dijadikan simpul-E yang baru.
- Pencarian dihentikan apabila telah menuju *goal node* yang diinginkan.

Berikut merupakan gambar ilustrasi algoritma *Backtracking* beserta skema umum algoritma *Backtracking* menggunakan rekursif:



Gambar 1. *Ilustrasi Algoritma Backtracking*

Skema Umum Algoritma *Backtracking* menggunakan rekursif :

```
Procedure backtracking (input k : integer)
{ Menentukan solusi dari persoalan algoritma backtracking }
Masukkan : k , yaitu indeks dari komponen solusi dari list, x[k]
Keluaran : Solusi x = {x[i],x[i+1],....x[n]}
Algoritma:
for tiap x[k] yang belum diexpand sehingga,
( x[k] <= T(k) ) and b(x[i],x[i+1],...x[n]) = true
do
        if (x[i],x[i+1],...x[n]) then
            cek(x)
        endif
            backtracking(k+1)
endfor
```

Knight's Tour

Knight's Tour adalah merupakan sebuah siklus hamilton, dimana langkah *knight* pada papan catur hanya melewat kotak 1 kali [4]. Aturan langkah *knight* pada papan catur adalah [5]:

- Melangkah dua persegi ke arah horizontal kemudian satu persegi ke arah vertikal atau,

- Melangkah dua persegi ke arah vertikal kemudian satu persegi ke arah horizotal, atau
- Melangkah satu persegi ke arah horizontal kemudian dua persegi ke arah vertikal
- Melangkah satu persegi ke arah vertikal kemudian dua persegi ke arah horizontal.

Dalam *Knight's Tour* dimana kuda melewati setiap kotak pada papan catur hanya 1 kali melawati dan kembali ke kotak semula maka disebut langkah kuda tertutup (*Closed Knight's Tour*). Sebaliknya, dimana semua kotak dilewati tidak kembali pada posisi semula maka disebut langkah kuda yang terbuka (*Open Knight's Tour*).

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Gambar 2. *Contoh Knight's Tour pada papan catur ukuran 8 x 8*

Gambar 3 di atas adalah contoh dari solusi *Knight's Tour*. Dimana angka 1-64 menandakan bahawa urutan kotak yang dapat dilewati oleh kuda. Langkah tersebut dapat diambil dari urutan pada papan catur dari kotak 1 hingga kotak 64. Pada penerapan *Knight's Tour* pada sirkuit Hamilton, terdapat 2 jalan dari *Knight's Tour* ini yaitu *closed tour* dan *open tour*. Berikut gambar *closed* dan *open tour*:

1	48	31	50	33	16	63	18
30	51	46	3	62	19	14	35
47	2	49	32	15	34	17	64
52	29	4	45	20	61	36	13
5	44	25	56	9	40	21	60
28	53	8	41	24	57	12	37
43	6	55	26	39	10	59	22
54	27	42	7	58	23	38	11

Gambar 3. *Open Knight's Tour*

18	59	50	1	48	15	22	63
51	2	17	60	21	64	47	14
58	19	4	49	16	23	62	45
3	52	57	20	61	46	13	24
34	5	40	53	36	25	44	11
39	56	35	8	41	12	29	26
6	33	54	37	28	31	10	43
55	38	7	32	9	42	27	30

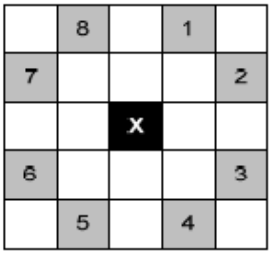
Gambar 4. *Close Knight's Tour*

HASIL DAN PEMBAHASAN

Implementasi Algoritma *Backtracking* pada *Knight's Tour*

Proses jalan *Knight's Tour* dalam permainan catur adalah membentuk huruf "L" pada setiap langkah di papan catur. Langkah-langkah yang dapat dilakukan oleh *Knight's Tour* adalah harus mengikuti aturan yang ada pada nilai offset ((2,1);(-2,1);(2,-1);(-2,-1);(1,2);(-1,2);(1,-2);(-1,-2)) dan nilai kolom maupun baris tidak melewati 8 yang mana papan catur 8 x 8, dijabarkan sebagai [6]:

1. A(x+2,y+1),
2. A(x-2,y+1),
3. A(x+2,y-1),
4. A(x-2,y-1),
5. A(x+1,y+2),
6. A(x-1,y+2),
7. A(x+1,y-2),
8. A (x-1,y-2).

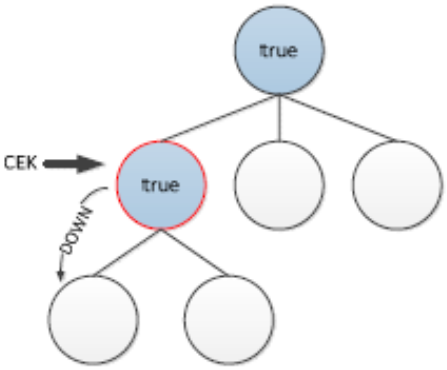


Gambar 5. *Langkah Knight*

Dari nilai offset yang didapat bahawa kemungkinan *knight* untuk melewati semua kotak pada papan catur hanya 1 kali dan tidak akan melewati kotak yang sama. Dimana papan catur yang digunakan adalah (m x n), m merupakan kolom dari papan catur dan n merupakan baris pada papan catur. Kemungkinan maksismum yang dilewati oleh knight adalah semua kotak papan catur berjumlah 64.

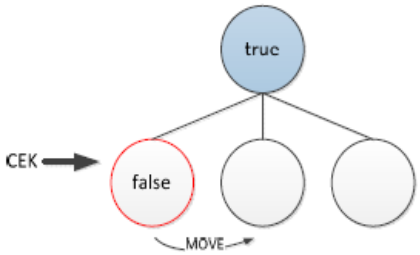
Langkah – langkah untuk mencari solusi dalam penggunaan algoritma *Backtracking* yaitu :

1. Langah pertama adalah menentukan titik awal dari *knight* dimana menjadikan sebagai node pertama atau disebut root.
2. Langkah berikut melakukan pengecekan pada root atau node pertama kali dibuat sebagai node yang valid, lalu telusuri anak dari node pertama yang akan di expand.
3. Selanjutnya lakukan fungsi yang mana tiap node diperluas hingga sampai simpul terdalam pastikan node tersebut valid.



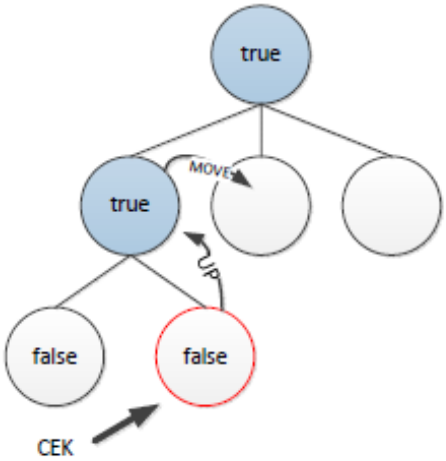
Gambar 6. *Expand node hingga bagian terdalam*

4. Jika pada node dicek tidak valid atau node sudah pada bagian terdalam tetapi tidak menemukan goal. Maka lakukan move ke sibling dari anak node yang tidak valid.



Gambar 7. *Move Sibling*

5. Proses berikutnya lakukan pengecekan setiap node yang telah di cek kevalidatannya didalam satu parent dan jika tidak ada lagi node dibawahnya yang bernilai valid. Maka pada kondisi tersebut lakukan proses *Backtacking* untuk naik ke node yang berikutnya agar dicek node sibling dari parent.



Gambar 8. *Backtracking terhadap node diatas pada parent*

Dengan melakukan proses di atas sampai menemukan goal dan dimana tidak melewati kotak yang sama pada papan catur, maka permasalahan dari *Knight's Tour* dapat terselesaikan dalam waktu yang lebih baik dibandingkan dengan menggunakan algoritma *brute force*. Proses pengecekan yang dilakukan dalam mengexpansi sebuah node dapat memudahkan implementasi algoritma *Backtracking* kedalam sebuah program. Pada proses expand maka diperlukan adjlist yang bertujuan sebagai

memberikan pengetahuan dari persoalan yang tidak diketahui jalur yang akan dilalui.

Berikut pseudo code implementasi algoritma *Backtracking* pada *Knight's Tour* dengan papan catur 8 x 8 :

```
Deklarasi:
  M : matriks [8][8] integer (M adalah
  matriks  8 kolom 8 baris)
  bar : integer ; kol : integer
Algoritma :
  for bar <= 0 to 7 do
     for kol <= 0 to 7 do
          M[bar][kol] <= 0
     endfor
 endfor
M[0][0] <= 1
while (bukan solusi) do
        if (lompat(M)) then
           maju( )
        endif
        else
           mundur( )
        endif
endwhile
```

Dalam penelitian ini papan catur di representasikan ke dalam sebuah matriks 8 x 8 dengan menggunakan matriks M[0][0] berisi angka 1 menandai bahwa posisi awal dari *knight*. Selanjutnya dilakukan pengulangan dimana akan dicek setiap node yang memungkinkan *knight* melewati kotak pada papan catur. Prosedur maju( ) akan melakukan pergerakkan terhadap *knight* dan memperbaruhi posisi matriks. Jika knight kehabisan ruang gerak dan node tidak valid maka dijalankan mundur( ) untuk memeriksa kemungkinan solusi lainya.

Implementasi Pada Program

Pada penelitian ini program yang digunakan untuk mencari solusi dari *Knight's Tour* dengan menggunakan bahasa pemograman python. Dalam program penjabarkan node menggunakan adjlist untuk menentukan child dari setiap node parent yang ada. Berikut tampilan dari program *Knight's Tour* yang dimulai titik awal adalah 1 :

```
========================
Papan Catur 1...64
========================
[1, 2, 3, 4, 5, 6, 7, 8]
[9, 10, 11, 12, 13, 14, 15, 16]
[17, 18, 19, 20, 21, 22, 23, 24]
[25, 26, 27, 28, 29, 30, 31, 32]
[33, 34, 35, 36, 37, 38, 39, 40]
[41, 42, 43, 44, 45, 46, 47, 48]
[49, 50, 51, 52, 53, 54, 55, 56]
[57, 58, 59, 60, 61, 62, 63, 64]
```

Gambar 9. *Papan Catur 8 x 8*

```
========================
Adj List 1...64
========================
1->[11, 18]
2->[12, 19, 17]
3->[13, 9, 20, 18]
4->[14, 10, 21, 19]
5->[15, 11, 22, 20]
6->[16, 12, 23, 21]
7->[13, 24, 22]
8->[14, 23]
9->[19, 3, 26]
10->[20, 4, 27, 25]
11->[21, 17, 5, 1, 28, 26]
12->[22, 18, 6, 2, 29, 27]
13->[23, 19, 7, 3, 30, 28]
14->[24, 20, 8, 4, 31, 29]
15->[21, 5, 32, 30]
16->[22, 6, 31]
17->[27, 11, 34, 2]
18->[28, 12, 35, 33, 3, 1]
19->[29, 25, 13, 9, 36, 34, 4, 2]
20->[30, 26, 14, 10, 37, 35, 5, 3]
21->[31, 27, 15, 11, 38, 36, 6, 4]
22->[32, 28, 16, 12, 39, 37, 7, 5]
23->[29, 13, 40, 38, 8, 6]
24->[30, 14, 39, 7]
25->[35, 19, 42, 10]
26->[36, 20, 43, 41, 11, 9]
27->[37, 33, 21, 17, 44, 42, 12, 10]
28->[38, 34, 22, 18, 45, 43, 13, 11]
29->[39, 35, 23, 19, 46, 44, 14, 12]
30->[40, 36, 24, 20, 47, 45, 15, 13]
31->[37, 21, 48, 46, 16, 14]
32->[38, 22, 47, 15]
```

```
33->[43, 27, 50, 18]
34->[44, 28, 51, 49, 19, 17]
35->[45, 41, 29, 25, 52, 50, 20, 18]
36->[46, 42, 30, 26, 53, 51, 21, 19]
37->[47, 43, 31, 27, 54, 52, 22, 20]
38->[48, 44, 32, 28, 55, 53, 23, 21]
39->[45, 29, 56, 54, 24, 22]
40->[46, 30, 55, 23]
41->[51, 35, 58, 26]
42->[52, 36, 59, 57, 27, 25]
43->[53, 49, 37, 33, 60, 58, 28, 26]
44->[54, 50, 38, 34, 61, 59, 29, 27]
45->[55, 51, 39, 35, 62, 60, 30, 28]
46->[56, 52, 40, 36, 63, 61, 31, 29]
47->[53, 37, 64, 62, 32, 30]
48->[54, 38, 63, 31]
49->[59, 43, 34]
50->[60, 44, 35, 33]
51->[61, 57, 45, 41, 36, 34]
52->[62, 58, 46, 42, 37, 35]
53->[63, 59, 47, 43, 38, 36]
54->[64, 60, 48, 44, 39, 37]
55->[61, 45, 40, 38]
56->[62, 46, 39]
57->[51, 42]
58->[52, 43, 41]
59->[53, 49, 44, 42]
60->[54, 50, 45, 43]
61->[55, 51, 46, 44]
62->[56, 52, 47, 45]
63->[53, 48, 46]
64->[54, 47]
```

Gambar 10. *Adjlist pada jalur yang akan dilalui Knight*

```
Jalur Knights Tour :
====================
[1, 18, 3, 20, 5, 22, 7, 24, 39, 54, 37, 52, 35, 50, 33, 27, 10, 25, 42, 57, 51,
 34, 17, 2, 19, 4, 21, 6, 23, 8, 14, 29, 12, 44, 59, 49, 43, 26, 9, 11, 28, 13,
30, 15, 32, 47, 62, 45, 60, 55, 38, 53, 36, 46, 31, 16, 48, 63, 61, 40, 56, 64,
41, 58]
```

Gambar 11. *Hasil jalur yang dilalui oleh Knight*

PENUTUP

Kesimpulan

Berdasarkan hasil yang dilakukan, maka dapat diambil kesimpulan bahwa algoritma *Backtracking* untuk mendapatkan solusi yang diberikan jauh lebih baik dibandingankan menggunakan algoritma *brute force*. Walaupun pencarian solusi yang diberikan dari algoritma *Backtracking* bukan yang paling optimum. Dengan menggunakan algoritma *Backtracking* memudahkan dalam permasalahan yang sering kali tidak memiliki informasi yang cukup untuk mengetahui pilihan selanjutnya. Oleh karena itu memudahkan merunut balik suatu langkah apabila menemui langkah terakhir atupun yang bukan langkah solusi sehingga solusi yang didapatkan menjadi lebih cepat.

Saran

Saran dalam pengembangan algoritma *Backtracking* pada permasalahan *Knight's Tour* maupun permasalahan lainnya, sebaiknya algoritma *Backtracking* dalam permasalahan apapun menambahkan fungsi heuristik atau metode lainnya. Sehingga keakuratan dalam mengambil keputusan lebih jauh baik dari sebelumnya. Dimana contoh dari fungsi heuristik bertujuan sebagai tolak ukur dalam pengambilan keputusan sehingga algoritma *Backtracking* menjadi lebih mangkus didalam permasalahan yang ada.

DAFTAR PUSTAKA

[1] . B. Hill and K. Tostado, "Knight ' s Tours," pp. 1–11, 2004.

[2] . A. Sulistyono, "Penggunaan Teori Graf dan Algoritma Backtrack dalam Penyelesaian Masalah Knight's Tour," Jur. Ilmu Komput., 2010.

[3] . R. Munir, "Algoritma Runut-balik (Backtracking)," in Strategi Algoritma, Bandung, 2018.

[4] . G. Lewandowski, "Project 1: The Knight's Tour Gary Lewandowski," CSCI 220 Fall, 2001.

[5] . E. Joni, "Analisis Teori Graf Pada Persoalan Knight??s Tour Dengan J2ME," J. TIMES, vol. 6, no. 2, pp. 52–57, 2017.

[6] . D. Yudhistira and D. Kurniasari, "Implementasi Algoritma Backtrack Untuk Pencarian Solusi Knight ' S Tour Problem Pada Papan Catur m x n," J. Komputasi, vol. 1, no. 1, pp. 43–53, 2012.