Metode Pencarian Lintasan Terpendek Dalam Graf

Edwin Romelta - 13508052

Jurusan Teknik Informatika , Institut Teknologi Bandung
Jalan Ganesha No. 10 Bandung
e-mail: edro_ooo@yahoo.co.id

ABSTRAK

Makalah ini membahas tentang beberapa metode pencarian lintasan terpendek pada graf. Terdapat beberapa cara untuk mencari lintasan terpendek dari suatu simpul dapa graf ke simpul lain. Dari beberapa cara itu tentu saja ada kekurangan dan kelebihannya. Semua itu tergantung dengan besar graf dan apa yang dibutuhkan dalam pencarian tersebut. Beberapa contoh dalam metoda pencarian lintasan terpendek pada graf adalah algoritma a*, algorima greedy, algoritma genetik hibrida, dan masih banyak lagi. Metode pencarian lintasan terpendek pada graf ini banyak digunakan dalam kihidupan. Banyak permasalahan yang perlu dipecahkan dengan metode ini. Seperti basisdata tersebar, pencarian jalan tol terpendek, dan lain – lain.

Dari banyak metode penyelesaian masalah lintasan terpendek pada graf, tentu saja banyak perbedaan mesikipun tujuan yang diharapkan sama. Yaitu menemukan lintasan terpendek / minimum yang optimum. Mesikipun tujuannya utamanya adalah mencari lintasan terpendek tetapi ada juga tujuan lainnya yaitu menemukan lintasan terpendek yang optimum dengan waktu pencarian yang minimum juga. Tetapi sangatlah sulit untuk mendapatkan kedua hal tersebut. Pencarian lintasan terpendek yang optimum tentu saja memakan waktu yang maksimum pula. Karena itulah banyak cara yang ditawarkan untuk pencarian lintasan terpendek pada graf dengan kelebihan dan kekurangan masing – masing. Kita tinggal memilih metode mana yang sesuai dengan tujuan utama dalam metode pencarian lintasan terpendek. Mencari lintasan terpendek yang optimum atau mencari lintasan terpendek dengan waktu yang minimum.

Kata kunci: graf, lintasan terpendek, algoritma

1. PENDAHULUAN

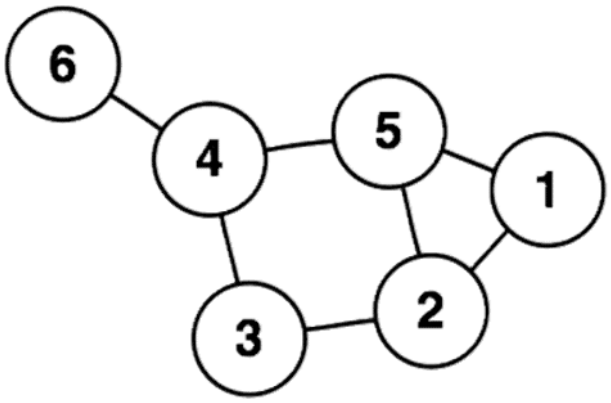
1.1 Graf

Graf adalah himpunan busur dan simpul yang banyaknya berhingga dan busur- busurnya menghubungkan sebagian atau keseluruhan pasangan dari simpul- simpulnya.

Graf G(V, E) terdiri atas himpunan simpul yang dinyatakan dengan V = {v1,v2, v3, ..., vn} dan himpunan busur yang dinyatakan dengan E = {e1, e2, e3, ..., en} dengan ei = (vi, vj) merupakan busur yang menghubungkan simpul vi dan simpul vj.

Dalam menggambarkan graf, simpul digambarkan dengan lingkaran kecil atau titik tebal dan busur digambarkan dengan garis, dan arah panah pada garis melambangkan arah dari garis tersebut. Nomor atau nama simpul dapat diletakkan di dalam lingkaran kecil atau di tepi titik tebal.

Busur (i,j) disebut busur berarah jika terdapat suatu aliran dari simpul i menuju ke simpul j. Dalam hal ini simpul i disebut simpul awal, sumber atau pangkal dan simpul j disebut simpul akhir, ujung, tujuan, atau terminal dari busur (i, j). Jika tidak terdapat aliran dari simpul i ke simpul j, maka busur (i, j) disebut busur tidak berarah.



Gambar 1. Graf dengan 6 node dan 7 arc

Banyak sekali struktur yang bisa direpresentasikan dengan graf, dan banyak masalah yang bisa diselesaikan dengan bantuan graf.

Jaringan persahabatan pada Facebook bisa direpresentasikan dengan graf: verteks-verteksnya adalah

para pemakai Facebook dan ada *edge* antara A dan B jika dan hanya jika A berteman dengan B.

Perkembangan algoritma untuk menangani graf akan berdampak besar bagi ilmu komputer.

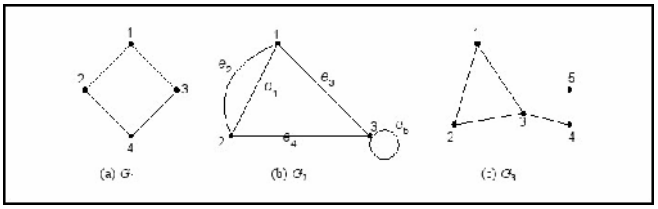
Sebuah struktur graf bisa dikembangkan dengan memberi bobot pada tiap *edge*. Graf berbobot dapat digunakan untuk melambangkan banyak konsep berbeda. Sebagai contoh jika suatu graf melambangkan jaringan jalan maka bobotnya bisa berarti panjang jalan maupun batas kecepatan tertinggi pada jalan tertentu.

Ekstensi lain pada graf adalah dengan membuat *edge*nya berarah, yang secara teknis disebut graf berarah atau digraf (*directed graph*). Digraf dengan *edge*berbobot disebut jaringan.

1.2 Lintasan

Lintasan yang panjangnya *n* dari simpul awal *v*0 ke simpul tujuan *vn* di dalam graf *G* ialah
barisan berselang-seling simpul-simpul dan sisi-sisi yang berbentuk *v*0, *e*1, *v*1, *e*2, *v*2,... , *vn* –1, *en*, *vn* sedemikian sehingga *e*1 = (*v*0, *v*1), *e*2 = (*v*1, *v*2), ... , *en* = (*vn*-1, *vn*) adalah sisi-sisi dari graf *G*.

Jika graf yang ditinjau adalah graf sederhana, maka kita cukup menuliskan lintasan sebagai barisan simpulsimpul saja: *v*0, *v*1, *v*2, ... , *vn* –1, *vn* , karena antara dua buah simpul berturutan di dalam lintasan tersebut hanya
ada satu sisi. Sebagai contoh, pada Gambar 2(a), lintasan 1, 2, 4, 3 adalah lintasan dengan barisan sisi
(1,2), (2,4), (4,3).



Gambar 2. Beberapa contoh graf

Pada graf yang mengandung sisi ganda, kita harus menulis lintasan sebagai barisan berselang-seling antara
simpul dan sisi menghindari kerancuan sisi mana dari sisi-sisi ganda yang dilalui. Misalnya pada Gambar 2(b),

1, *e*1, 2, *e*4, 3, *e*5, 3

adalah lintasan dari simpul 1 ke simpul 3 yang melalui sisi *e*1, *e*4, dan *e*5.

Catatlah bahwa simpul dan sisi yang dilalui di dalam lintasan boleh berulang. Sebuah lintasan dikatakan

lintasan sederhana (*simple path*) jika semua simpulnya berbeda (setiap sisi yang dilalui hanya satu kali).

Lintasan yang berawal dan berakhir pada simpul yang sama disebut lintasan tertutup (*closed path*),
sedangkan lintasan yang tidak berawal dan berakhir pada simpul yang sama disebut lintasan terbuka (*open path*).

Pada Gambar 2(a),
lintasan 1, 2, 4, 3 adalah lintasan sederhana, juga lintasan terbuka.
lintasan 1, 2, 4, 3, 1 adalah juga lintasan sederhana, juga lintasan tertutup.
lintasan 1, 2, 4, 3, 2 bukan lintasan sederhana, tetapi lintasan terbuka.

Panjang lintasan adalah jumlah sisi dalam lintasan tersebut. Lintasan 1, 2, 4, 3 pada Gambar 2(a)
memiliki panjang 3.

Lintasan terpendek adalah jalur yang dilalui dari suatu node ke node lain dengan besar atau nilai pada sisi yang jumlah akhirnya dari node awal ke node akhir paling kecil.

Lintasan terpendek adalah lintasan minimum yang diperlukan untuk mencapai suatu tempat dari tempat lain.

Lintasan minimum yang dimaksud dapat dicari dengan menggunakan graf. Graf yang digunakan adalah graf yang berbobot, yaitu graf yang setiap sisinya diberikan suatu nilai atau bobot. Dalam kasus ini, bobot yang dimaksud berupa jarak dan waktu kemacetan terjadi.

Ada beberapa macam persoalan lintasan terpendek, antara lain:
a. Lintasan terpendek antara dua buah simpul tertentu (*a pair shortets path*).
b. Lintasan terpendek antara semua pasangan simpul (*all pairs shortest path*).
c. Lintasan terpendek dari simpul tertentu ke semua simpul yang lain (*single-source shoertest path*).
d. Lintasan terpendek antara dua buah simpul yang melalui beberapa simpul tertentu (*intermediate shortest path*).

Dalam makalah ini, persoalan yang digunakan adalah *single-source shortest path*.

2. METODE

2.1 Algoritma Greedy

PSEUDECODE Algoritma Greedy

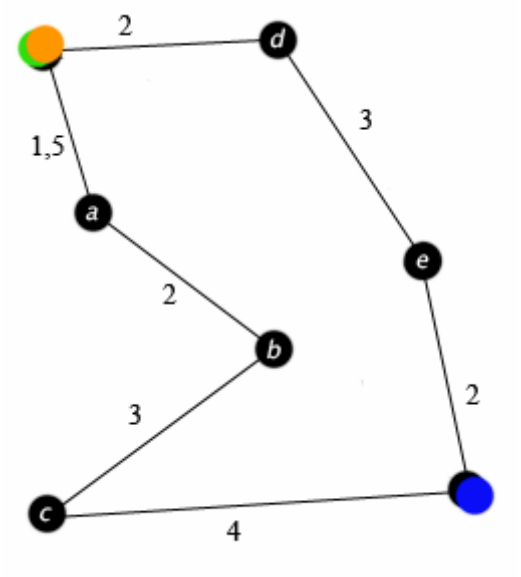
```
set Greedy (Set Candidate){
  solution= new Set( );
  while (Candidate.isNotEmpty()) {
          next = Candidate.select(); //use
selection criteria,
           //remove from Candidate and return
value
          if (solution.isFeasible( next))
//constraints satisfied
                        solution.union( next);
                        if (solution.solves())
return solution}
  //No more candidates and no solution
  return null
}
```

algoritma greedy memiliki kecepatan pencarian yang lebih singkat dibandingkan algoritma a* karena ia hanya membandingkan jarak dari tempat asal dengan tempat lain.

Tetapi algoritma greedy ini memiliki optimasi pencarian lintasan terpendek yang kurang baik karena hal tersebut.

Jika ditengah perncarian dia hanya menemukan suatu simpul dan padahal simpul tersebut tersebut jaraknya sangat besar , algoritma greedy tetap memilih jalur tersebut karena tidak dapat kembali lagi ke awal dan tidak ada pilihan lain.

Sehingga pencarian tidak optimal, hal yang lebih buruknya adalah gagal dalam pencarian. Seperti kasus menukar uang 41dolar dengan pecahan 25,10,4. pencarian akan gagal dengan uang 25,10,4 dan sisa 2 dolar. Yang seharusnya 25 dan 4 buah 4 dolar.



Gambar 3. contoh graf

Contoh Persoalan :
"Carilah jalur tenpendek dari titik kuning ke titik biru"

Pilihan awal yang dipilih algoritma adalah a karena a lebih pendek daripada d.

Pilihan selanjutnya hanya satu sehingga tidak ada pilihan lain selain b. Lalu ke c dan ke tujuan akhir.

Maka jaraknhya adalah 10,5
Padahal jika menggunakan jalur satu lagi sebesar 7

Begitu seterusnya

Jika jarak a ke b adalah 1000. algoritma ini tidak bisa mundur, sehingga memilih b. Padahal nilainya sangat besar. Disanalah kelemahan algoritma ini.

Tetapi dengan tidak pernah mundur ke tempat awal untuk mencari jalan alternatif. Algoritma ini cepat dalam menyelesain pencarian lintasan tercepat.

2.2 Algoritma A*

Berbeada denagn algoritma greedy. Algoritma ini menghitung semua kemungkinan dan menyimpannya sehingga jika setiap memilih jalan. Ia juga membandingkan dengan jalan lain yang disimpan .
Sehingga hasil pencarian lintasan tercapat dengan mengunakan algoritma ini akan menghasilkan hasil yang optimum.

Namun karena ia terus membandingkan algoritma ini memakan waktu yang cukup lama. Sehingga jika simpulnya sangat banyak akan memakan waktu yang sangat lama.

PSEUDOCODE ALGORITMA A*

```
function A*(start,goal)
    closedset := the empty set                % The set of nodes already evaluated.
    openset := set containing the initial node % The set of tentative nodes to be evaluated.
    g_score[start] := 0                       % Distance from start along optimal path.
    h_score[start] := heuristic_estimate_of_distance(start, goal)
    f_score[start] := h_score[start]          % Estimated total distance from start to goal through y.
    while openset is not empty
        x := the node in openset having the lowest f_score[] value
        if x = goal
            return reconstruct_path(came_from,goal)
        remove x from openset
        add x to closedset
        foreach y in neighbor_nodes(x)
            if y in closedset
                continue
            tentative_g_score := g_score[x] + dist_between(x,y)

            if y not in openset
                add y to openset

                tentative_is_better := true
            elseif tentative_g_score < g_score[y]
                tentative_is_better := true
            else
                tentative_is_better := false
            if tentative_is_better = true
                came_from[y] := x
                g_score[y] := tentative_g_score
                h_score[y] := heuristic_estimate_of_distance(y, goal)
                f_score[y] := g_score[y] + h_score[y]
    return failure

 function reconstruct_path(came_from,current_node)
    if came_from[current_node] is set
        p = reconstruct_path(came_from,came_from[current_node])
        return (p + current_node)
    else
        return the empty path
```

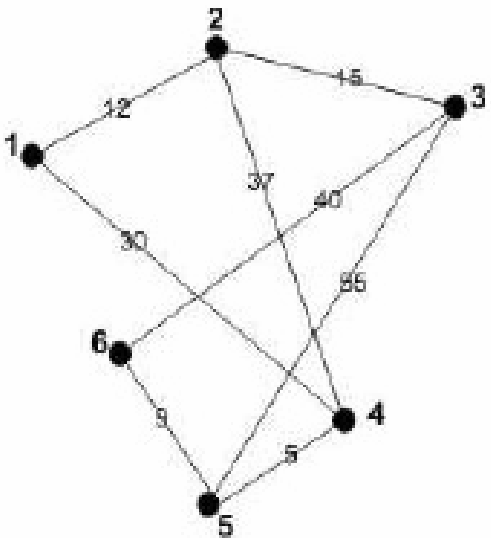
Contoh penerapan algoritma a*:

Contoh Persoalan pada gambar 3:
"Carilah jalur tenpendek dari titik kuning ke titik biru"

Pilihan awal yang dipilih algoritma adalah a karena a lebih pendek daripada d.

Disimpanlah
Jarak1 (a)= 1,5
Jarak2 (d)= 2
Dipilihlah jarak 1 karena lebih pendek
Jarak1 (a b) = 3,5
Jarak2 (d) = 2
Dipilihlah jarak 2 karena lebih pendek
Jarak1 (a b) = 3,5
Jarak2 (d e) = 5
Dipilihlan jarak 1 karena lebih pendek
Jarak1 (a b c) = 6,5
Jarak2 (d e) = 5
Dipilihlan jarak 2 karena lebih pendek
Jarak1 (a b c) = 6,5
Jarak2 (d e) = 7
Meskipun telah sampai tujuan tetapi jalur lain belum mencapai tujuan sehingga pencarian masi dilanjutkan
Jarak1 (a b c) = 10,5
Jarak2 (d e) = 7
Setelah seluruh jalur sampai pada tujuan barulah dibandingkan untuk terakhir kalinya jalur mana yang paling optimum
Karena jalur 2 lebih pendek maka dipilihlah jarak 2.

Untuk melakukan pencarian pada graf yang kecil seperti gambar 3 saja algoritma ini sudah memakan waktu yang sangat banyak. Padahal graf tersebut hanya memiliki 2 jalur. Bayangkan jika graf tersebut dipersulit sedikit seperti gambar 4



Gambar 4. Contoh Graf

Jika diminta mencari jarak dari 2 ke 5 akan memakai banyak jalur.
Dari tempat awal saja sudah menjadi 3 jalur.
Belum lagi di simpul – simpul lainnya yang bercabang – cabang.

Tetapi dari keseluruhan algoritma ini menghasilkan hasil yang optimal tetapi mengunakan waktu yang maksimal pula.

Kompleksitas Waktu Asimptotik dari algoritma a* adalah

$$| h(x) - h^{*}(x) | = O(\log h^{*}(x))$$

2.3 Algoritma Genetik

Algoritma genetika (*Genetic Algorithm*, GA) adalah algoritma pencarian yang didasarkan atas mekanisme seleksi alami dan evolusi biologis. Algoritma genetika mengkombinasikan antara deretan struktur dengan pertukaran informasi acak ke bentuk algoritma pencarian dengan beberapa perubahan bakat pada manusia. Pada setiap generasi, himpunan baru dari deretan individu dibuat berdasarkan kecocokan pada generasi sebelumnya

PSEUDECODE Algoritma Genetik Hibrida

```
begin GA
  g:=0  { generation counter }
  Initialize population P(g)
  Evaluate population P(g)  { i.e.,
compute fitness values }
  while not done do
    g:=g+1
    Select P(g) from P(g-1)
    Crossover P(g)
    Mutate P(g)
    Evaluate P(g)
  end while
end GA
```

Pada algoritma genetika terdapat beberapa proses yaitu:

a. Proses Pengkodean (*Encoding*)
Pada proses pengkodean, gen dapat direpresentasikan dalam bentuk string bit, pohon, array bilangan real, daftar aturan, elemen permutasi, elemen program, atau representasi lainnya yang dapat diimplementasikan untuk operator genetika.

b. Proses Seleksi
Seleksi adalah proses untuk menentukan individu mana saja yang akan dipilih untuk dilakukan rekombinasi dan bagaimana keturunan terbentuk dari individu-individu terpilih tersebut

c. Proses Rekombinasi
Rekombinasi adalah proses untuk menyilangkan dua kromosom sehingga membentuk kromosom baru yang harapannya lebih baik dari pada induknya

d. Proses Mutasi
Mutasi adalah proses penambahan nilai acak yang sangat kecil dengan probabilitas rendah pada variabel keturunan.

Cara kerja algoritma genetika sangat sederhana, hanya mencakup proses penduplikasian string-string dan pertukaran bagian-bagian dari string. Meskipun cukup sederhana, tetapi mempunyai kemampuan untuk menyelesaikan persoalan optimasi. Kemampuan ini didukung oleh tiga operator genetik yaitu reproduksi, rekombinasi dan mutasi. Pada reproduksi terjadi proses penduplikasian string berdasarkan nilai fungsi objektifnya. Nilai objektif ini dapat dilihat sebagai suatu keuntungan yang ingin dicapai atau dimaksimalkan. Sementara proses pertukaran bagian-bagian string dilakukan oleh operator rekombinasi dan mutasi.

Di samping ketiga operator dasar (reproduksi, rekombinasi, dan mutasi), parameter-parameter genetik (jumlah populasi, maksimum generasi, probabilitas rekombinasi, probabilitas mutasi, dan lain-lain), serta asumsi-asumsi yang digunakan dalam pemodelannya juga mempunyai peran penting.

3. KESIMPULAN

Pencarian lintasan terpendek sangat berguna karena banyak kasus dalam kehidupan ini yang dapat diselesaikan dengan metode tersebut.

Metode – metode pencarian lintasan terpendek memiliki kelebihan dan kekurangan masing – masing. Kita harus memilihnya sesuai dengan kebutuhan permasalahan yang kita hadapi.

Untuk mengukur performansi metode pencarian, terdapat empat kriteria yang dapat digunakan, yaitu :

- Completeness : Apakah metode tersebut menjamin penemuan solusi jika solusinya memang ada?
- Time Complexity : Berapa lama waktu yang diperlukan?
- Space complexity : Berapa banyak memori yang diperlukan?
- Optimality : Apakah metode tersebut menjamin menemukan solusi yang terbaik jika terdapat beberapa solusi berbeda?

Tingkat persentase optimasi algoritma greedy cukup rendah. Namun memiki waktu yang cepat dibanding dengan metoda lain.

Metode greedy dapat dipakai jika kita lebih mementingkan kecepatan dalam pencarian lintasan tercepat.

Metode greedy dapat dipakai jika jarak pada antar simpul tidak jauh berbeda dibandingkan dengan jarak antar simpul lain. Dan dapat digunakan juga jika cabang simpul banyak.

Metode a* memiliki optimasi yang sangat tinggi tetapi mengunakan waktu yang sangat lama pula untuk mencari hasil dari pencarian tersebut.

Jika kita lebih mementingkan pencarian yang lebih optimal dan kita memiliki waktu yang cukup lama algoritma a* lah jawabannya.

Tingkat persentase optimasi algoritma genetika ditentukan oleh maksimum generasi. Semakin banyak generasi yang diproses, maka semakin tinggi tingkat optimasi suatu kasus

REFERENSI

[1] [Munir, Rinaldi. 2005, Matematika Diskrit. Bandung: Penerbit Informatika.
[2] http://journal.uii.ac.id/index.php/Snati/article/viewFile/163/1406
[3] http://en.wikipedia.org/wiki/Shortest_path_problem
[4] http://en.wikipedia.org/wiki/A*_search_algorithm
[5] http://en.wikipedia.org/wiki/Genetic_algorithm
[6] http://en.wikipedia.org/wiki/Greedy_algorithm
[7] http://repository.gunadarma.ac.id:8000/Kommit2000_komputasi_002_261.pdf
[8] repository.gunadarma.ac.id:8000/Kommit2000_komputasi_008_191.pdf