LESSON 5 :

INFORMED SEARCH

Part I

3.1 Pengantar

Kita telah menunjukan beberapa metoda pencarian yang berbeda. Di bagian bagian awal bab ini kita telah menunjukan beberapa metode pencarian buta (blind search). Metoda pencarian tanpa informasi seperti ini(uninformed search method) tidak memiliki pengetahuan khusus tentang permasalah yang dihadapinya sehingga metode metode ini tidak efisien untuk banyak kasus. Menggunakan informasi khusus yang spesifik untuk suatu masalah tertentu akan sangat memperbaiki kecepatan pencarian solusi, Di bab ini kita akan mempelajari beberapa algoritma pencarian dengan informasi (informed search algorithm) yang menggunakan heuristic yang spesifik pada masalah yang dihadapinya.

Review beberapa Strategi pencarian

1. Pencarian buta (blind search)
    a. Depth First Search
    b. Breadth First Search
    c. Itterative Deepening Search
    d. Bidirectional Search
2. Pencarian dengan informasi (informed search)

3.1.1. Algoritma pencarian graf

Kita mulai dengan melihat algoritma pencarian graf

```
                Graph search algorithm
Let fringe be a list containing the initial state
Let closed be initially empty
Loop
    if fringe is empty return failure
    Node ← remove-first (fringe)
                if Node is a goal
                   then return the path from initial state to Node
      else put Node in closed
                      generate all successors of Node S
                      for all nodes m in S
                if m is not in fringe or closed
                            merge m into fringe
End Loop
```

3.1.2. Review untuk uniform cost-search

Kita akan mereview sebuah variasi breadth-first search yang telah lebih dulu kita deskripsikan di bahan yang lalu

Dalam Uniform cost search kita mengurutkan node berdasarkan biaya yang ditempuhnya

Bila g(n) = biaya jalur dari node awal ke node saat ini yaitu node ke n

Algoritma ini mengurutkan node berdasarkan nilai g secara membesar. Dan membuka node yang dengan biaya paling kecil pada fringe

Properti Unifrom cost Search

- Complete
- Optimal/ admissible
- Waktunya exponensial
- Space complexitynya : linear / O(bd)

Algoritma ini menggunakan nilai g(n) untuk memilih node yang akan dikembangkan/ di expand. Kita sekarang akan memperkenalkan metode pencarian dengan informasi / pencarian dengan heuristik yang menggunakan informasi heuristic yang specific sesuai dengan masalah yang dihadapi. Heuristic inilah yang akan digunakan untuk memilih node yang akan dikembangkan / diexpand.

3.1.3 Pencarian dengan informasi (informed search)

Kita dapat amati bahwa metode pencarian tanpa informasi yang mencari solusi di dalam state space secara sistematis tidak efisien dalam banyak kasus. Pencarian dengan informasi menggunakan informasi yang spesifik dari permasalahan yang bersangkutan. Dan karenanya mungkin menjadi lebih efisien. Inti dari algoritma seperti ini adalah konsep tentang fungsi heuristik

3.1.3.1 Heuristik

Heuristik sendiri berarti “rule of thumb”. Seperti yang dikatakan Judea Pearl . (Heuristic are critarea, methods, or principle for deciding which among several alternatives course of action promises to be the most effective in order to reach some goal) “Heuristik adalah suatu kritarea, metode, atau prinsip dalam memilih antara beberapa kemungkinan aksi yang dianggap adalah aksi aksi yang dapat mencapai tujuan dengan paling efisien.” Dalam pencarian heuristic atau pencarian dengan informasi, heuristik dapat digunakan untuk mengindentifikasikan jalur pencarian yang paling menjanjikan.

Contoh fungsi heuristik

Fungsi heuristik pada node ke n adalah perkiraan biaya optimum dari node ini ke tujuan, dilambangkan dengan notasi h(n).

h(n) = perkiraan biaya terkecil dari node ke n ke node tujuan (goal)

Contoh 1: kita mencari jalur dari kalkuta ke guwahati. Maka salah satu heuristik untuk guwahati bisa berupa jarak garis lurus (straight line distance) antara kalkuta dan guwahati

h(kalkuta) = euclidianDistance (kalkuta, guwata)

Contoh 2 : 8-puzzle :

Heuristik misplaced tile : jumlah angka yang salah tempat

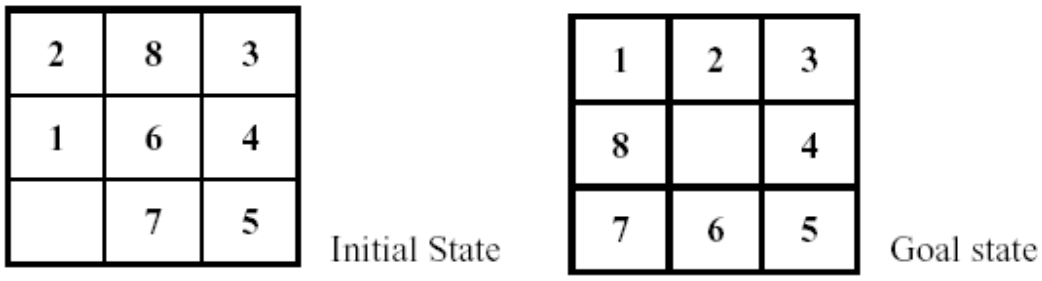


Figure 1: 8 puzzle

Gambar pertama menunjukan state n , dan gambar kedua menunjukan state tujuan h(n) = 5 karena angka 2,8,1,6,7 salah tempatnya

Heuristik Manhatan distance : salah satu heuristic untuk 8-puzzle adalah manhattan distance. Heuristik ini menjumlah jarak dari angka yang salah tempat. Jarak diukur dari jumlah perbedaan suatu angka dengan posisi angka sebenarnnya dalam baris dan kolomnya.

h(n) = 1 + 1 + 0 + 0 + 0 + 1 + 1 + 2 = 6

Kita sekarang akan mempelajari satu pencarian heuristik, Best-first Search

3.2 Best First Search

Uniform cost search adalah kasus khusus dari algoritma ini. Algoritma ini tetap memliki priority queue dari node yang akan diperiksa. Suatu fungsi biaya (f(n)) diterapkan pada setiap node. Node kemudian ditempatkan terurut berdasarkan nilai fnya pada queue, node dengan nilai f yang paling kecil adalah nilai yang paling kecil kemudian akan diproses terlebih dahulu. Bentuk umum algoritma best first search adalah :

```
Best First Search
Let fringe be a priority queue containing the initial state
Loop
    if fringe is empty return failure
    Node ← remove-first (fringe)
                if Node is a goal
                   then return the path from initial state to Node
      else generate all successors of Node, and
        put the newly generated nodes into fringe
        according to their f values
End Loop
```

Kita akan melihat beberapa cara berbeda dalam menentukan fungsi f. Ini menghasilkan algoritma pencarian yang berbeda.

3.2.1 Greedy search

Pada greedy search ide utamanya adalah mengembangkan node dengan nilai estimasi biaya ke goal yang paling kecil (berarti node yang paling dekat ke tujuan).

Kita menggunakan fungsi heuristik

ffn) = h(n)

h(n) = perkiraan jarak yang tersisa ke goal
algoritma ini seringkali berhasil dengan sangat baik. Algoritma ini cenderung menemukan jawaban dengan cepat walau tidak selalu solusi yang optimal,.

Algoritma ini tidak optimal, dan selain itu tidak juga complete. Algoritma ini dapat gagal menemukan solusi walaupun solusi itu ada. Hal ini bisa dilihat bila kita menjalankan algoritma greedy pada contoh di bawah ini : . Heuristik yang baik untuk permasalahan mencari rute adalah garis lurus ke tujuan

Figure 2 is an example of a route finding problem. S is the starting state, G is the goal state.

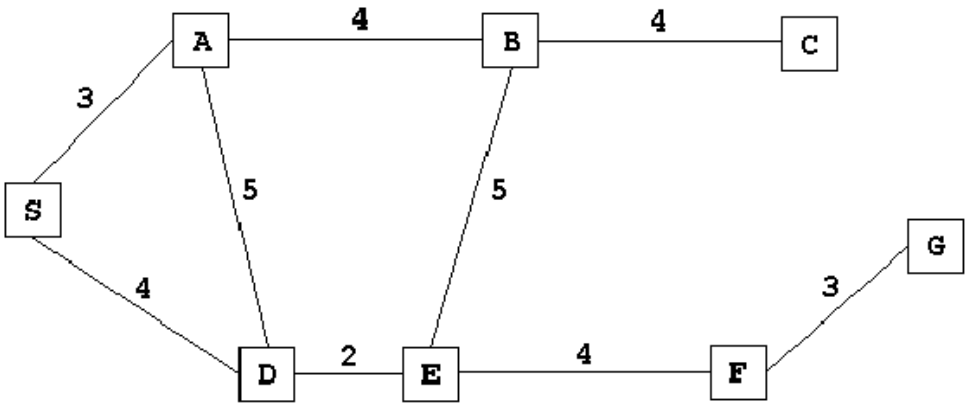


Figure 2

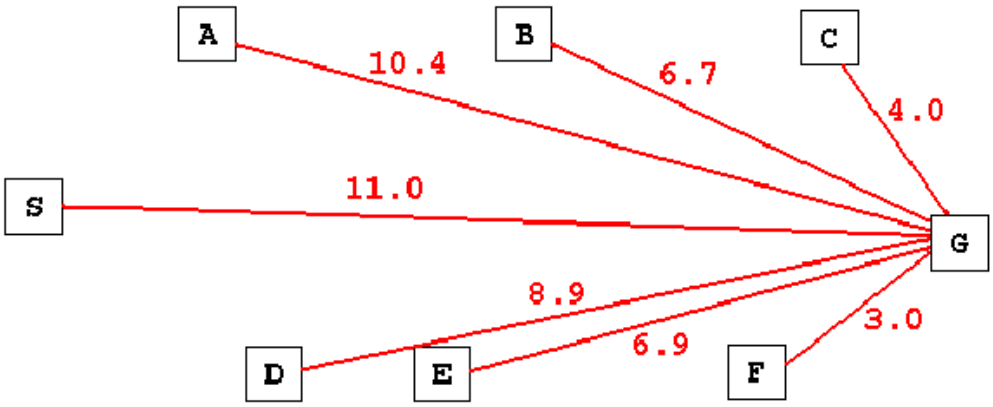
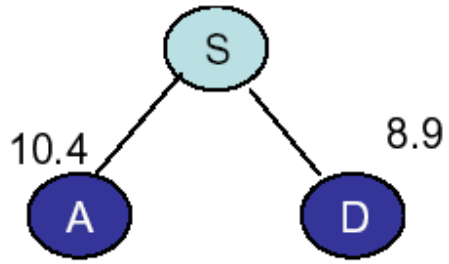


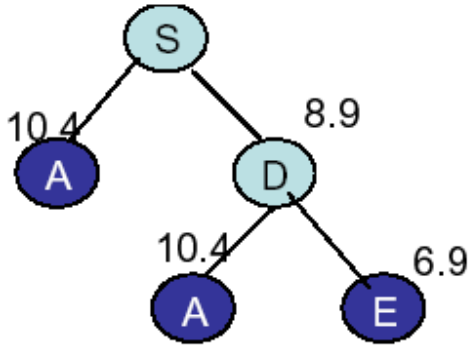
Figure 3

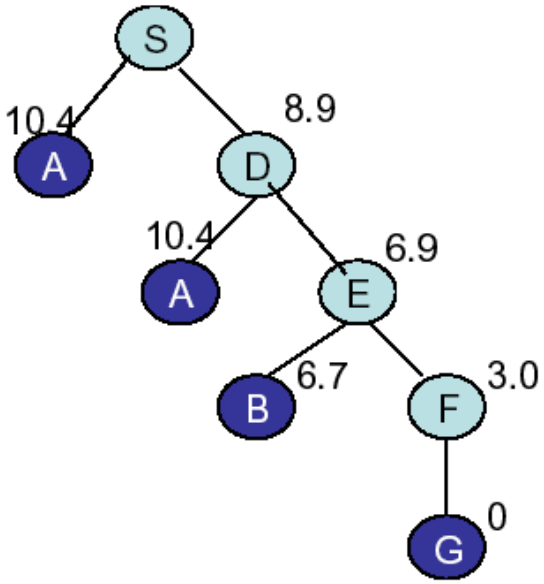
Mari kita jalankan algoritma greedy search untuk graf pada gambar ke 2. Heuristik jarak ditunjukan oleh gambar 3 .

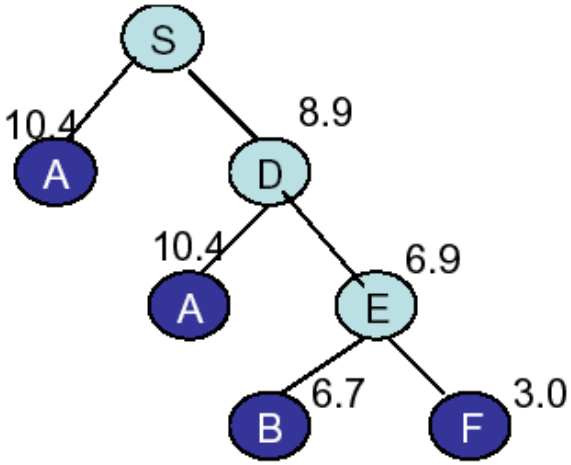
Langkah 1 : S di buka, node turunannya adalah D dan A



Langkah 2 : Karena nilai D lebih kecil ia diexpand terlebih dahulu




S
10.4
A
8.9
D
10.4
A
6.9
E
6.7
B
3.0
F
0
G


S
10.4
A
8.9
D
10.4
A
6.9
E
6.7
B
3.0
F

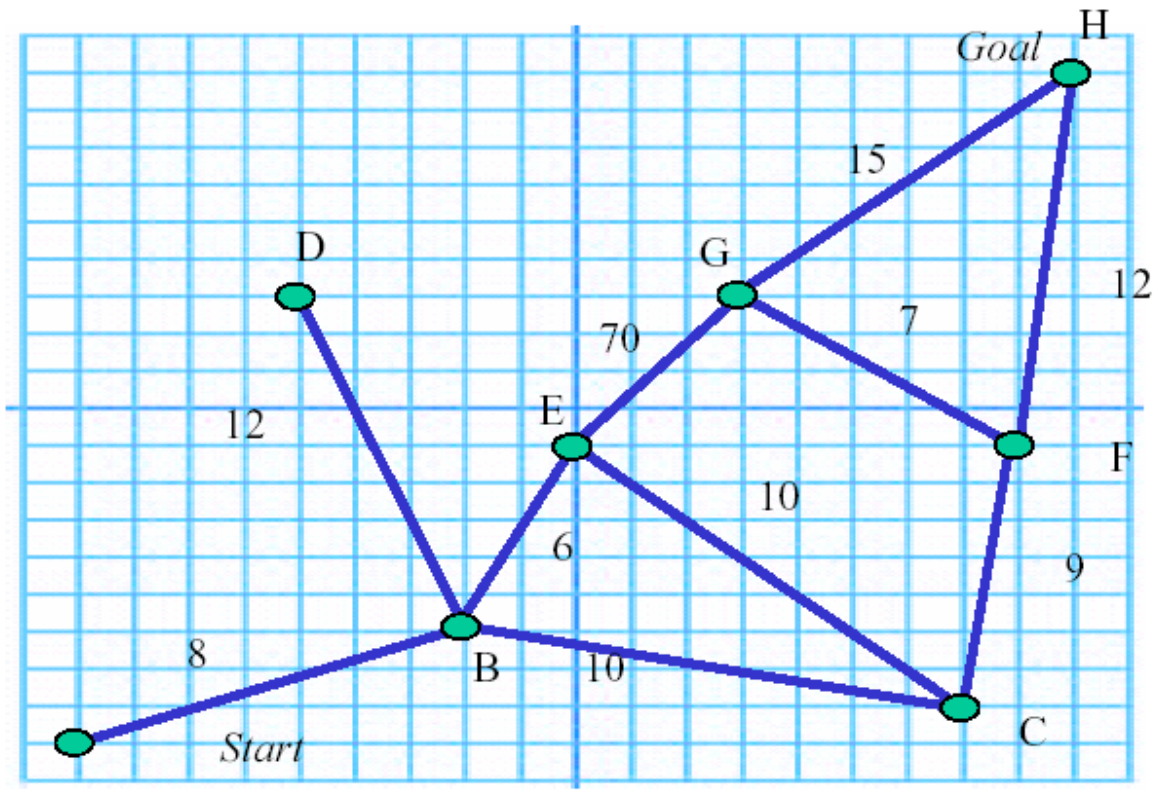


Figure 4

Ilustrasi Greedy Search

Kita akan menjalankan algoritma greedy search pada masalah di gambar 4, Kita gunakan heuristik garis lurus. h(n) adalah jarak garis lurus dari n ke goal .

Maka node yang akan diexpand adalah :

- A
- B
- E
- G
- H

Jalur yang akan diambil adalah jalur A-B-E-G-H dengan biaya 99, jelas bukan jalur yang optimal, jalur A-B-C-E-H hanya perlu biaya 39

3.2.2. A* Search

Berikutnya kita akan melihat algoritma A* yang terkenal. Algoritma ini diperkenalkan oleh Hart Nilsson dan Rafael di 1968.

A* adalah salah satu algoritma best first search dengan

f(n) = g(n) + h(n)

dimana :

g(n) adalah jumlah biaya dari node awal ke node n

h(n) adalah perkiraan biaya yang dibutuhkan dari n ke tujuan

f(n) adalah jumlah jarak yang sudah ditempuh sekarang + perkiraan jarak yang tersisa

h(h) dikatakan dapat diterima (admissible) bila ia menaksir terlalu rendah (underestimate) biaya dari setiap solusi yang dapat dicapai dari n. bila C*(n) adalah biaya dari solusi yang termurah dari n ke tujuan dan jika h(n) admissible maka

h(n) <= C*(n)

Jika kita dapat membuktikan bahwa h(n) admissible, maka pencarian akan dapat menemukan solusi yang optimal

Secara garis besar algoritma A* adalah sebagai berikut :

```
Algorithm A*
OPEN = nodes on frontier.     CLOSED = expanded nodes.
OPEN = {<s, nil>}
while OPEN is not empty
    remove from OPEN the node <n,p> with minimum f(n)
    place <n,p> on CLOSED
    if n is a goal node,
          return success (path p)
    for each edge connecting n & m with cost c
       if <m, q> is on CLOSED and {p|e} is cheaper than q
          then remove n from CLOSED,
                put <m,{p|e}> on OPEN
       else if <m,q> is on OPEN and {p|e} is cheaper than q
          then replace q with {p|e}
       else if m is not on OPEN
          then put <m,{p|e}> on OPEN
return failure
```

3.2.1 ilustrasi A*

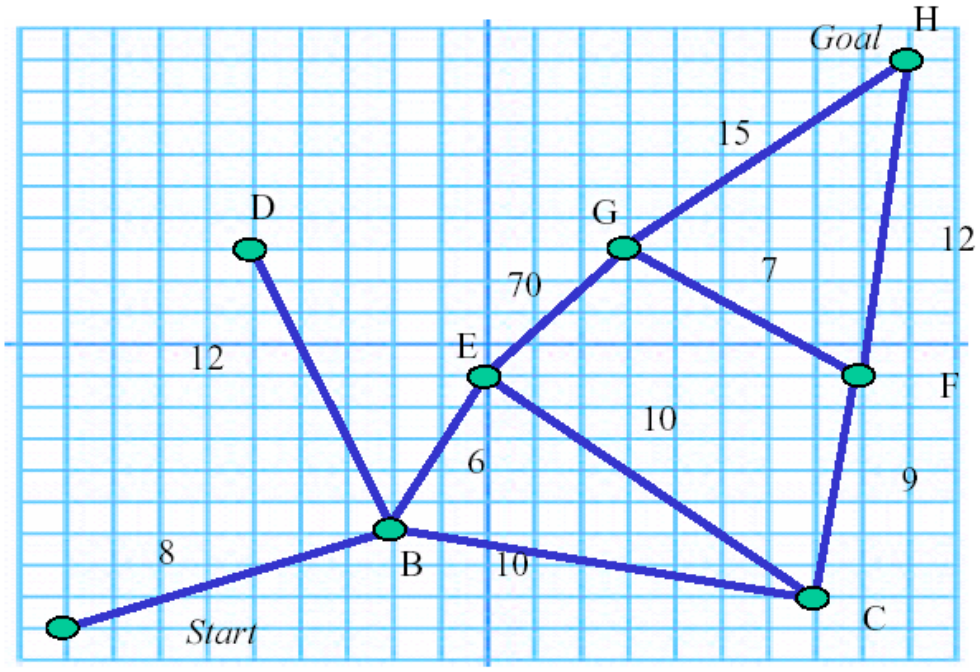


Figure 4

Heuristik yang digunakan adalah straight line distance (garis lurus). Urutan node yang akan dibuka dan status dari fringe dapat ditunjukan oleh tabel berikut :

Steps	Fringe	Node expanded	Comments
1	A		
2	B(26.6)	A	
3	E(27.5), C(35.1), D(35.2)	B	
4	C(35.1), D(35.2), ~~C(41.2)~~ G(92.5)	E	C is not inserted as there is another C with lower cost.
5	D(35.2), F(37), G(92.5)	C	
6	F(37), G(92.5)	D	
7	H(39), G(42.5)	F	G is replaced with a lower cost node
8	G(42.5)	H	Goal test successful.

The path returned is A-B-C-F-H.
The path cost is 39. This is an optimal path.

3.2.2 Properti pencarian A*

Algoritma A* bersifat admissible. Ini berarti apabila solusi ada, solusi yang ditemukan pertama adalah solusi yang optimal. A* bersifat admissible bila memenuhi syarat syarat berikut :

- Di dalam graph state space
  - Setiap node memiliki successor yang terbatas
  - Setiap arc pada graph memiliki biaya yang > dari 0
- Heuristik : untuk setiap node n h(n) < h*(n)

A* juga complete bila memenuhi syarat syarat diatas:

A* optimal untuk heuristik tertentu – dari semua algoritma search yang optimal yang mencari dari root ke node tujuan, dapat dibuktikan bahwa tidak ada algoritma lain yang membuka simpul node yang paling sedikit dan menemukan solusi

Namun, jumlah node yang disimpan masih bersifat exponential pada kasus terburuk

Sayangnya perkiraan saja tidak cukup bagi algoritma A* untuk tidak membuka simpul secara exponential, sebagai tambahan A* harus menyimpan semua node pada memory

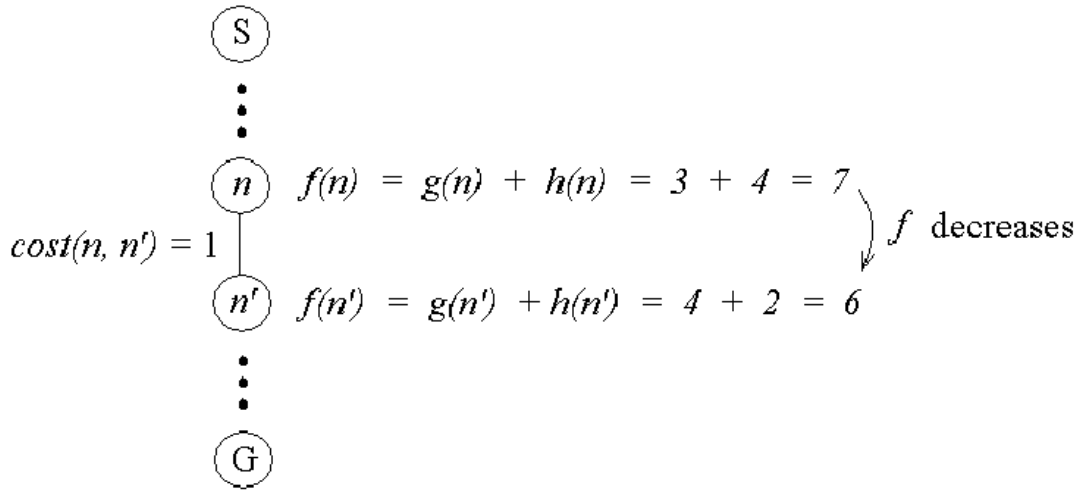
A* jauh lebih efisien daripada pencarian tanpa informasi

Selalu lebih baik menggunakan heuristic yang bernilai lebih besar asalkan tidak melebihi nilai aslinya (overestimate).

Suatu heuristik dikatakan konsisten bila :

$h(n) <= cost(n,n') + h(n')$

sebagai contoh heuristik untuk kasus dibawah ini tidak konsisten, karena h(n)= 4 sedangkan h(n') = cost(n,n') + h(n') = 1 + 2 = 3, sehingga h(n) > h(n'). hal ini membuat nilai f mengecil dari node n ke node n' :



Jika suatu heuristik h konsisten, maka nilai f sepanjang jalur mana pun tidak akan menurun

f(n')

= perkiraan jarak dari awal ke tujuan melalui n'

= jarak yang telah ditempuh dari awal ke n +biaya jarak dari n ke n' + perkiraan jarak tersisa dari n' ke tujuan

= g(n) + cost (n,n') + h(n')

>= g(n) + h(n) karena cost(n,n') + h(n') > h(n)

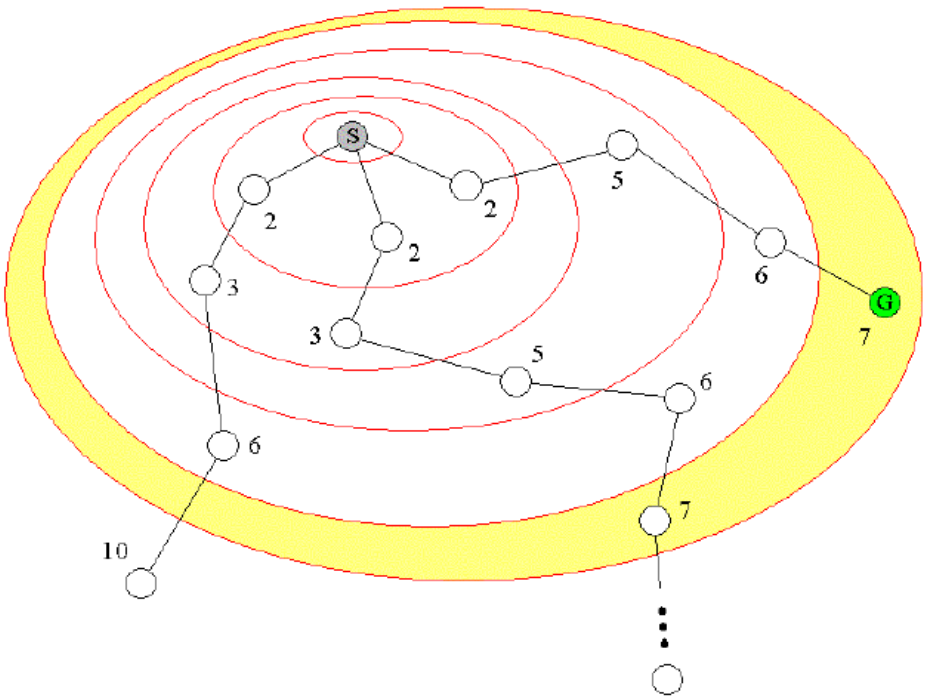
= f(n)

Karena itu f(n') > = f(n), jadi f seharusnya tidak pernah berkurang di sepanjang jalur

Bila heuristik h tidak konsisten, kita dapat memperbaiki nilai f agar seakan akan nilai h konsisten menggunakan persamaan pathmax

f(n') = max(f(n), g(n') + h(n'))

ini menjamin nilai f tidak akan mengecil sepanjang jalur dari awal ke goal. Bila diberikan nilai f yang tidak menurun, kita dapat menganggap A* sebagai mencari dari luar melalui serangkaian kontur node, dimana tiap node di dalam suatu kontur memiliki nilai f yang sama



Untuk setiap kontur, A* memeriksa semua node pada kontur yang sama , jika solusi ada, node tujuan terdekat dengan node awal akan diketemukan pertama kali.

Kita sekarang akan membuktikan admissibility dari A*

3.2.3 bukti admissibility dari A*

Kita akan menunjukan bahwa algoritma A* admissible bila menggunakan heuristic yang monotone Heuristik yang monotone adalah heuristik yang untuk setiap jalur kearah tujuan nilai f nya tidak mengecil . Seperti yang telah disebutkan sebelumnya bila heuristik tidak monotone dapat diubah terlebih dahulu menjadi heuristik yang monotone dengan menggunakan trik ini (m adalah anak dari n)

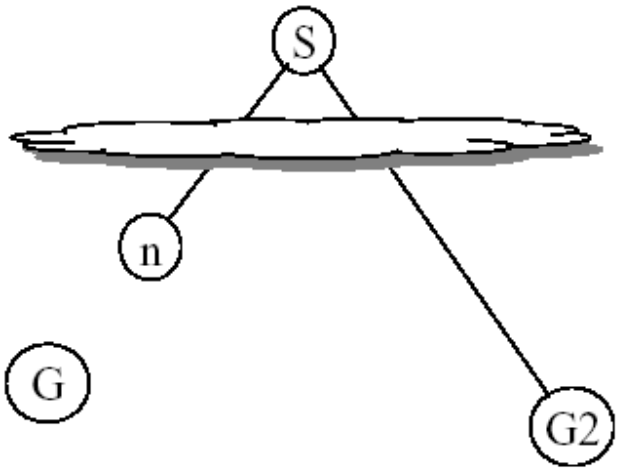
f(m) = max(f(n),g(m) + h(m))

dengan G adalah state tujuan yang optimal

C* adalah nilai biaya jalur yang optimal

G2 adalah state tujuan yang lebih tidak optimal dengan g(G2) > C*

Anggaplah A* memilih G2 dari proses OPEN untuk dikembangkan / diekspansi



Anggaplah ada sebuah node n sebuah jalur yang menuju pada state tujuan yang optimal G, maka C* >= f(n). Karena n tidak dipilih untuk diexpansi bila dibandingkan dengan G2 maka f(n) >= f(G2)

Karena G2 adalah sebuah state tujuan maka f(G2) = g(G2)

Maka C* > = g(G2)

Ini adalah suatu kontradiksi. Maka A* tidak akan memilih G2 untuk diekspansi sebelum mencapai tujuan dengan jalur yang optimal.

3.2.4 Bukti kelengkapan algoritma A*

Bila ada sebuah G yang adalah state tujuan yang optimal.

A* tidak bisa mencapai state tujuan hanya bila terdapat node sejumlah tidak terbatas dimana f(n) <= C*

Ini hanya akan terjadi apabila

Terdapat node yang mempunyai branching factor tidak terbatas

Terdapat jalur yang memiliki biaya yang terbatas tapi memiliki jumlah node yang tidak terbatas. Tapi kita telah mengasumsikan bahwa untuk setiap arc pada graf memiliki biaya yang lebih besar dari 0. Oleh karena itu apabila ada node yang jumlahnya tidak terhingga pada jalur g(n) < f*, maka biaya jalur itu akan jadi tidak terbatas

Maka A * dikatakan complete dan optimal dengan mengasumsikan sebuah heuristic yang admissible dan konsisten (atau menggunakan pathmax untuk mempertahankan konsistensi)

A* juga dikatakan optimally efficient berarti bahwa algoritma ini akan mengembangkan hanya node node yang diperlukan saja untuk menjamin optimalitas dan efisiensi

3.2.4 Analisis performansi A*

Modelkan search space dengan sebuah pohon b-ary yang uniform dengan state awal yang unik s dan state tujuan g, g ada sejauh N dari s.

Jumlah node yang diekspansi oleh A* berjumlah eksponensial sebesar N kecuali apabila perkiraan heuristiknya akurat secara logaritmik

$|h(n)-h^*(n)| <= O (\log h^*(n))$

Pada prakteknya kebanyakan heuristic mengalami proportional error

Terkadang menjadi sulit untuk menggunakan algoritma A* karena queue nya menjadi terlalu besar

Solusinya adalah menggunakan algoritma yang memakan memory lebih sedikit.

3.2.5 properti dari heuristik

Dominansi

h2 dikatakan memiliki dominansi atas h1 jika $h2(n) >= h1(n)$ untuk setiap n

A * akan mengekspansi lebih sedikit node dengan h2 secara rata rata dibandingkan dengan h1

Bukti :

Untuk setiap node dimana $f(n) < C^*$ akan diekspansi, maka n akan diekspansi apabila

$h(n) < f^* - g(n)$

karena $h2(n) >= h1(n)$ setiap node yang diekspansi h1 akan diekspansi oleh h1.

3.2.6. Menggunakan Heuristik lebih dari satu

Apabila anda menemukan sejumlah heuristik yang non – overestimating untuk sebuah masalah :

h1(n),h2(n),h3(n),…hk(n)

ama

max (h1(n),h2(n),….hk(n))

adalah sebuah heuristik yang non –overestimating yang efektif. Ini bisa dibuktikan dengan properti dominansi