Hill Climbing dan Least-cost

wijanarto

IDE

- HCA memilih langkah selanjutnya pada node yang muncul untuk di tempatkan di tempat *terdekat* dari tujuan. (sehingga , dia merupakan bagian terjauh dari posisi saat ini).
- Meminimalkan jumlah koneksi dalam graph

Properti Hill Climbing

- *Hill-climbing* algorithms (*gradient descent* jika fungsi evaluasi mewakili cost) hanya mempertimbangkan state saat ini. Semua state sebelumnya di lupakan.
- Hill-climbing merupakan resemble depth-first search sebab dia cenderung bergerak cepat ke bawah satu jalur.
- ``Pure'' hill-climbing tidak mendukung backtracking jadi tiap langkah adalah irrevocable.
- Initial state di pilih secara random! . Al*gorithm mungkin memberi solusi yang berbeda tergantung pada awal state*

Properti Hill Climbing

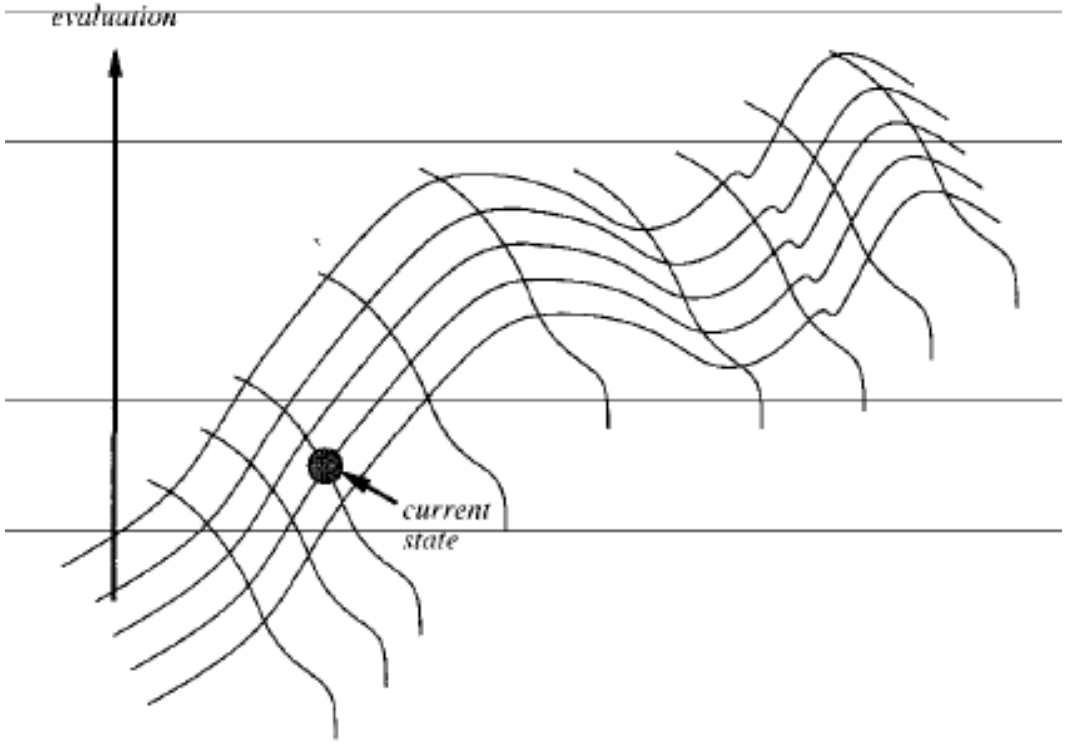
- *HCA secara sederhana melakukan loop yang dilanjutkan dengan bergerak dalam arah nilai yang menaik. Algoritma tidak mengelola tree,*
- *Sehingga struktur data node diperlukan hanya untuk mencatat state dan mengEVALUASI nya , yang di notasiikan dengan VALUE.*
- *Jika terdapat lebih dari satu solusi di suksesor untuk di pilih maka algoritma dapat memilihnya secara random, dengan beberapa pertimbangan seperti terdapat pada **fenomena HCA**.*

Algoritma 1

```
function HiLL-CLiMBiNG(problem)
            returns a solution state
inputs: problem, a problem
static: current, a node ; next, anode
Current← MakeNode(initial-state[problem])
loop do
   next ← a highest-valued successor of current
   if VALUE[next] < VALUE[current] then return
  current
   current ←next
end
```

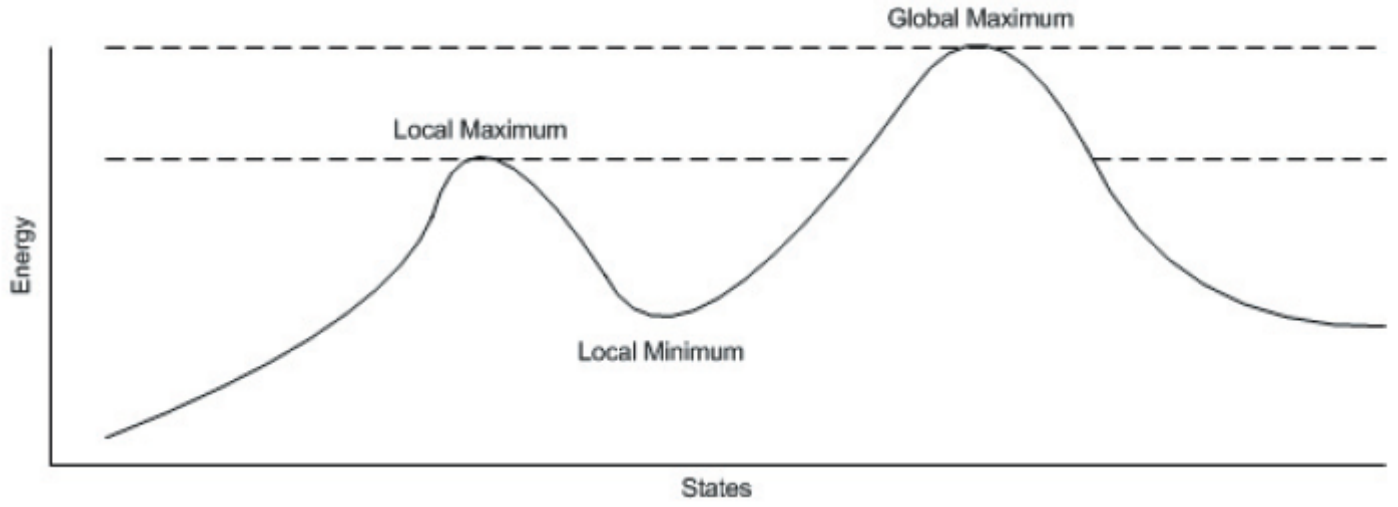
HCA model 1

- Algoritma secara iterative mencari peak pada permukaan state dimana tingginya didefnisikan oleh fungsi evaluasi



HCA model 2

- Ada local optimum dan global optimum. Tujuannya utk me-maximalkan function, tapi jika kita mulai dari kiri dan bergerak kekanan ke arah global optimum, kita akan dapatkan jalan buntu pada local optimum.
- Masalah HCA : node terbaik utk di enumaerasi scr lokal mkn bukan node global yang terbaik
- Sehingga HCA dapat dimulai dari lokal optimum, untuk mendapatkan global optimum (solusi terbaik yang ada).



Fenomena HC

- **Local Maxima**: puncak lokal yang bujan merupakan puncak tertinggi secara global.
- **Plateaus**: Daerah dari state space yang datar (semua state punya ukuran evaluasi yang sama secara virtual).
- **Ridges (1):** keniscayaan untuk mencapai slope lebih tinggi dalam satu langkah ; harus bergerak turun dulu dalam rangka bergereak ke atas.
- **Ridges (2):** Area dengan langkah slop pada bawah, tapi dekat dengan slope atas.

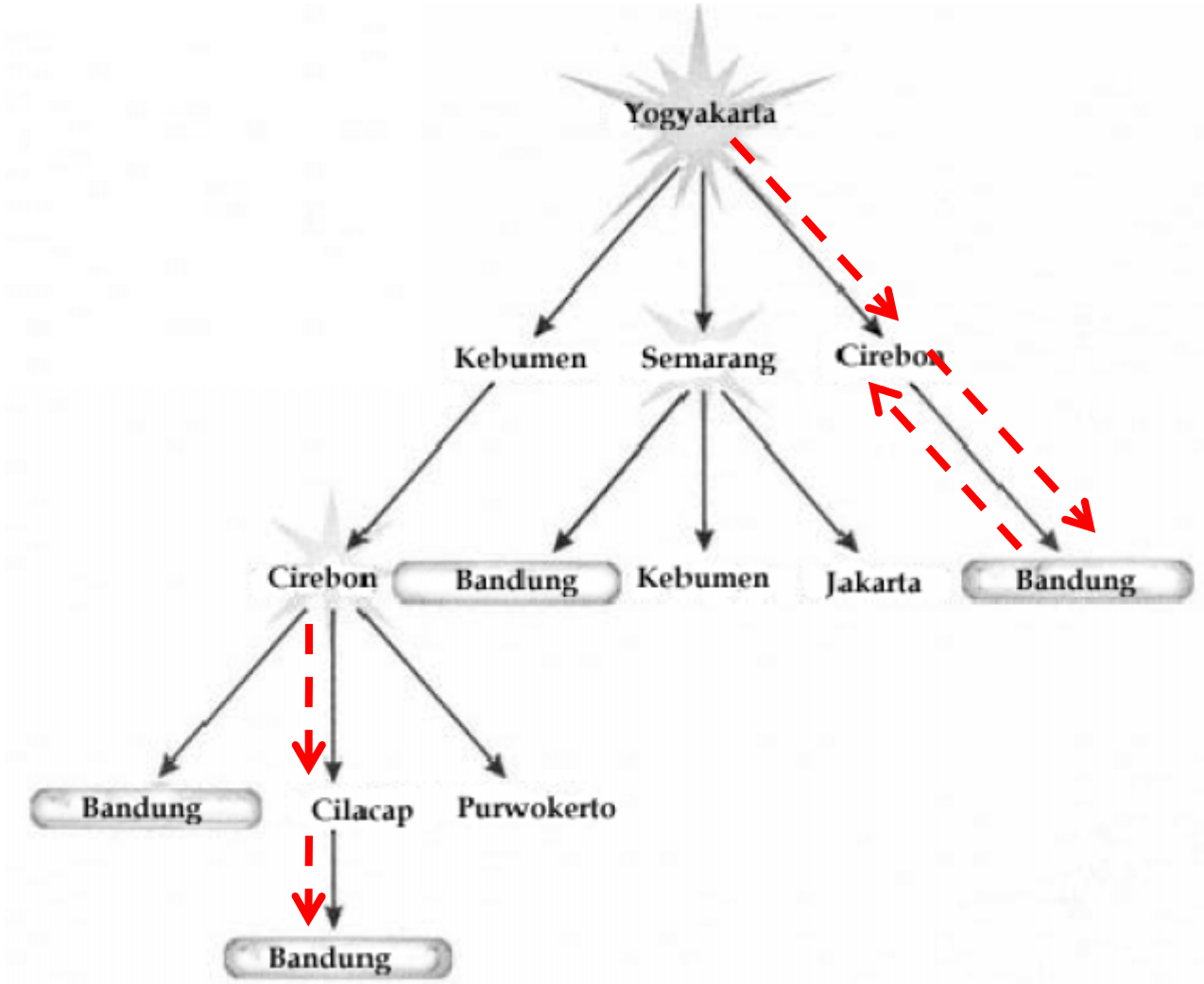
Variasi HC

- **Random restart**: Menjalankan HCA beberapa kali dengan inisial state random . Solusi terbaik yang aman.
- **Add backtracking**: Melihat seluruh successor dalam list terurut untuk pertimbangan masa datang (next step)
- **Jumping:** Melompat ke bagian baru dari ruang pencarian jika menemui jalan buntu.

Algoritma 2

```
function hill-climbing(node)
   node = select-random-node;
   loop
      if (node is goal) then return(node);
      successors = generate-successors(node);
      best-successor = select-best-node(successors);
      if (value(best-successor) < value(node))
         then return(node);
         node = best-successor;
    end loop
```

Yogyakarta – Cirebon - Bandung
Jarak = 2900



Contoh HCA

```
// max. no. state tetangga dari state yang di berikan
#define MAX_NEIGHBOUR   100
// asumsi fungsi external  (utk tiap domain masalah)
extern double Cost(STATE *pS);
// mengembalikan cost dari state yag diberikan
extern int Solution(STATE *pS);
// 1 jika state adalah solusi; else 0
// generate state tetangga dari state yang diberikan;
// mengembalikan no. state tetangga
extern int GetNeighbours(STATE *pS, STATE Next[MAX_NEIGHBOUR]);
```

Contoh HCA dalam C

```
// Cari dan temukan lowest cost state
// STATE : representasi problem dari struktur  state
// pS0 dan pS mrpk pointer yg diberikan sbg init state
// dan low cost state yang ditemukan.
int HillClimbing(STATE *pS0, STATE *pS)
{
   STATE Next[MAX_NEIGHBOUR];
   int i, n, index;
   double c0, c, c1;
   if ( Solution(pS0) ) {// ketemu
       CopyState(pS,pS0);
       return(1);
   }
   CopyState(pS, pS0); // init
   c0 = Cost(pS0);     // cost of initial state
```

Contoh HCA

```
while ( (n = GetNeighbours(pS,Next)) > 0 )
        // cari tetangga s
   {
        index = 0;
        // index (dlm Next) dari tetangga lowest cost
        c = Cost(&Next[0]);
        for(i = 0; i < n; i++)
         // lakukan utk tiap state tetangga
        {
        if ( Solution(&Next[i]) ) // ketemu
        {
            CopyState(pS,&Next[i]);
            return(1);
        }
```

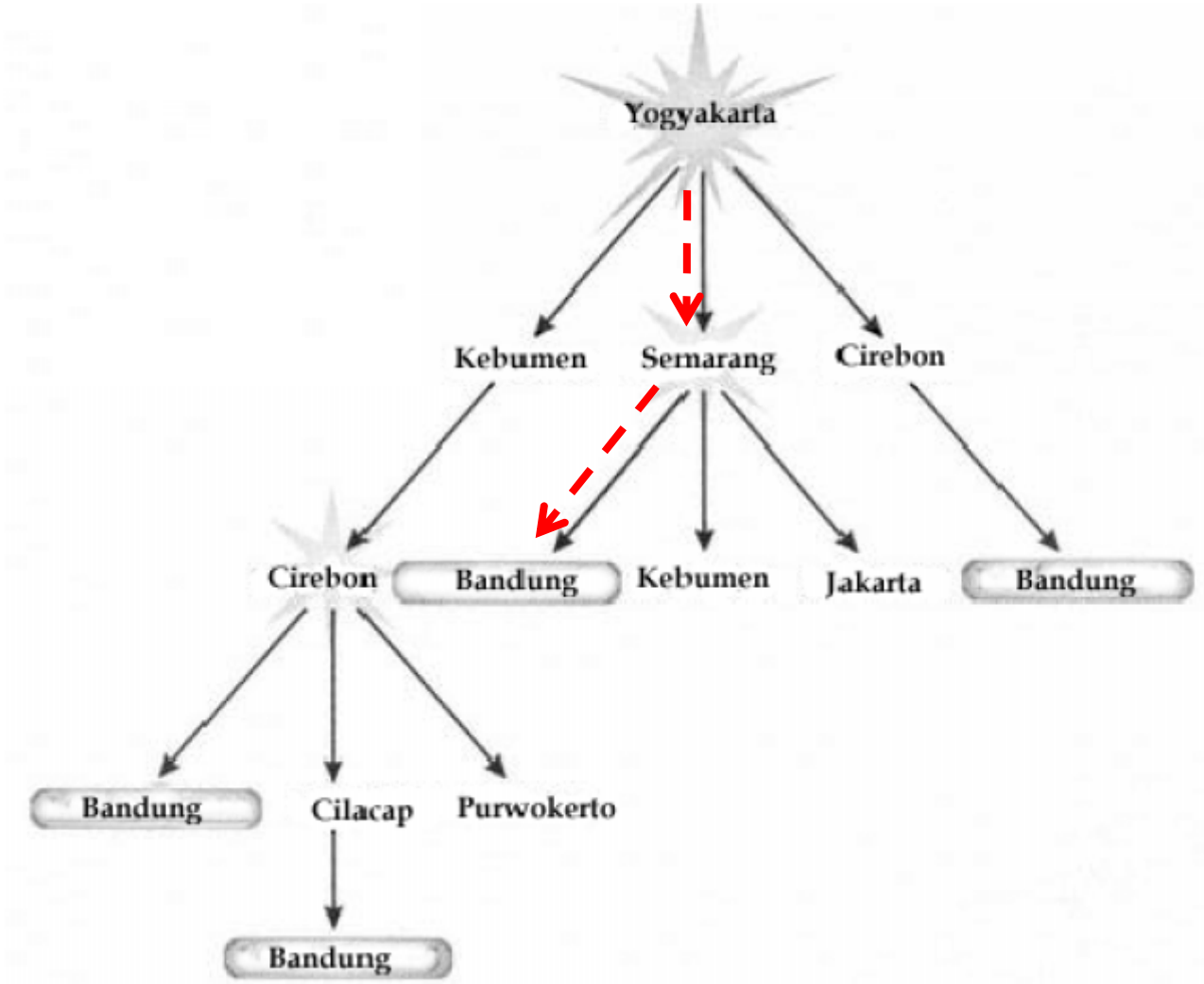
Contoh HCA

```
if ( (c1 = Cost(&Next[i])) < c )
        {
            index = i;
            c = c1;
        }
        } // end for
        if ( c < c0 ) // ketemu tetangga dg lower cost
        {
        CopyState(pS, &Next[index]);
        c0 = c;
        }
        else // mencapai local maximum
        break;
    } // end while
    return(0);
  // solusi global tdk ditemukan; S= menemukan low cost state
}
```

Least-Cost

- Kebalikan dari hill-climbing
- Mengambil jalur dengan sedikit mungkin resistensinya.
- Mencari minimal cost dalam tiap path

Tree Least Cost



OPTIMAL SOLUTION

- Simulated Annealing
- Best First Search
- Z*, A*, AO, dan AO*
- Implementasi dalam c, kecuali optimal solution dapat di lihat di wijanarto.wordpress.com