LESSON 6 :

INFORMED SEARCH

Part II

3.3 Itterative deepening A* search

3.3.1 Algoritma IDA*

Itterative deepening search atau IDA* serupa dengan iterative deepening depth first, namun dengan modifikasi sebagai berikut :

Batas kedalaman digantikan dengan batas nilai f

1. Mulailah dengan flimit = h(start)
2. Potonglah (prune) semua node dimana f(node) > flimit
3. flimit berikutnya adalah nilai minimum node yang dipotong

saat node yang terpotong diekspansi tergantung pada nilai f dari node tersebut.

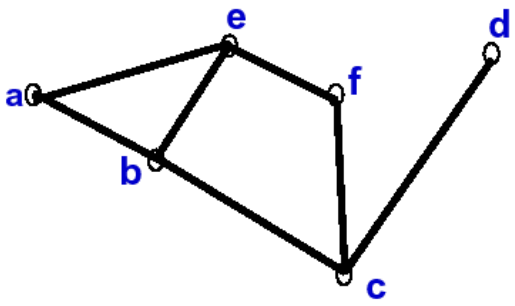


Figure 1

Perhatikan graf pada gambar diatas, pada iterasi pertama hanya node a yang diekspansi, ketika a diekspansi b dan c dihasilkan, dan setelah dievaluasi maka nilai f node b dan node c adalah 15

Untuk iterasi berikutnya flimit berikutnya adalah 15 dan dalam iterasi ini a, b dan c diekspansi, ini dilukiskan oleh gambar dibawah ini

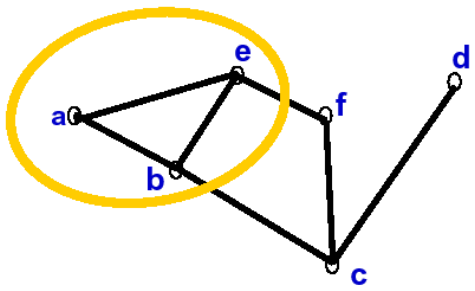


Figure 2: f-limit = 15

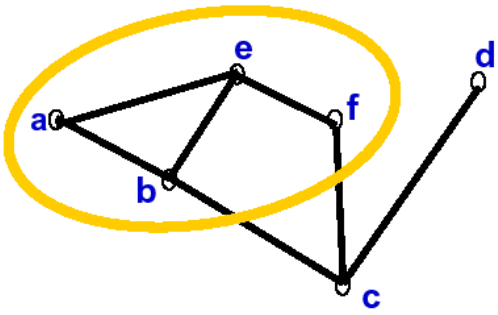


Figure 3: f-limit = 21

3.3.2 Analisis algoritma IDA*

IDA* complete dan dengan optimal menggunakan tempat memory secara linear. Tiap iterasinya adalah depth first search dan karenanya tidak membutuhkan priority queue

Jumlah node yang dibuka relatif terhadap A * adalah tergantung pada jumlah nilai heuristic yang unik. Jumlah iterasi sama dengan jumlah nilai f yang unik yang kurang atau sama dengan C*

Dalam permasalahan seperti 8 puzzle menggunakan heuristic manhattan distance, ada sedikit kemungkinan nilai f (nilai f hanya bilangan bulat). Karenanya jumlah node yang diekspansi mendekati jumlah yang diekspansi oleh algoritma A*

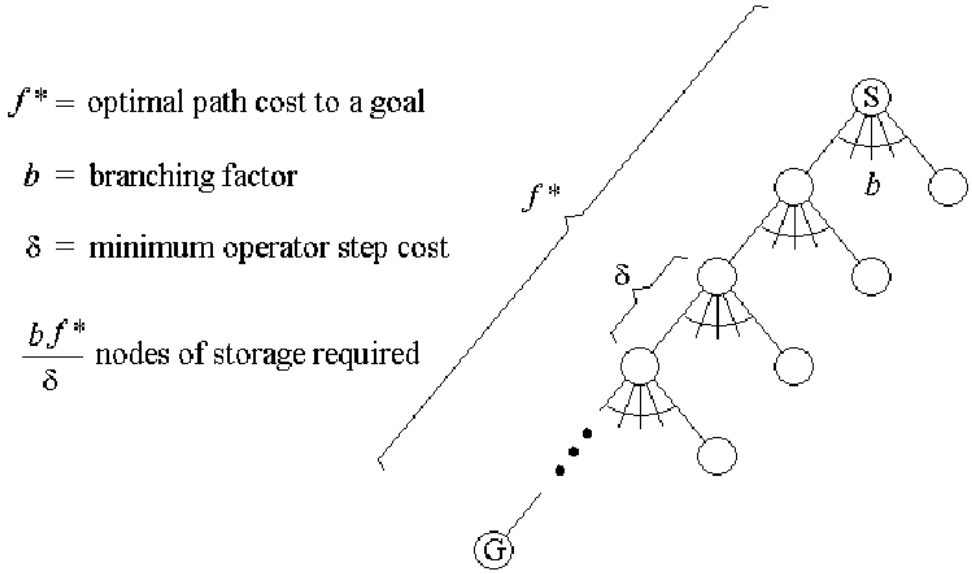
Namun dalam permasalahan seperti Travelling salesman problem (TSP) yang menggunakan nilai real. Setiap nilai f mungkin unik. Dan karenannya banyak node yang harus diekspansi. Dalam kasus yang paling buruk, apabila semua nilai f unik, algoritma ini akan mengekspansi hanya 1 node untuk tiap iterasi, sehingga apabila A* menghasilkan N node, jumlah maksimum node yang akan dihasilkan oleh IDA* adalah 1 + 2 + … + N =O (N2)

Kalau seperti itu mengapa menggunakan IDA* ? Biasanya bila menggunakan A*, untuk permasalahan yang lebih besar algoritma ini kehabisan memory lebih dulu dibandingkan kehabisan waktu. IDA* dapat digunakan pada masalah masalah seperti ini karena keperluan memorynya linear bahkan 15-puzzle yang tidak dapat diselesaikan A* karena kehabisan memory dapat diselesaikan dengan IDA*.

IDA * jelas tidak cocok untuk permasalahan seperti travelling salesman problem.IDA* juga menghasilkan node yang berulang pada graf dengan siklus, strategi pencarian depth first tidak cocok untuk graf yang memiliki banyak siklus.

Kebutuhan tempatnya : O(bd)

IDA* bersifat complete, optimal, dan efisien (dengan asumsi menggunakan heuristic yang konsisten dan admissible), dan IDA* hanya membutuhkan tempat penyimpanan yang polynomial pada kasus terburuknya.



3.4 Algoritma pencarian dengan memory terbatas lainnya

IDA * menggunakan sangat sedikit memory

Algoritma lain dapat menggunakan memory lebih banyak untuk pencarian yang lebih efisien

3.4.1 RBFS : Recursive Breadth First Search

RBFS menggunakan kebutuhan memory yang linear

Cara kerjanya meniru algoritma best first search

RBFS menyimpan nilai f terbaik dari jalur alternatif yang tersedia dari pendahulu node yang sekarang

Bila node yang sekarang melebihi limit nilai f maka jalur alternatif lainnya diperiksa.

RBFS mengingat nilai f daun terbaik di sub pohon yang sudah dilalui

```
RBFS (node: N, value: F(N), bound: B)

   IF f(N)>B, RETURN f(N)
   IF N is a goal, EXIT algorithm
   IF N has no children, RETURN infinity
   FOR each child Ni of N,
         IF f(N)<F(N), F[i] := MAX(F(N),f(Ni))
         ELSE F[i] := f(Ni)
sort Ni and F[i] in increasing order of F[i]
   IF only one child, F[2] := infinity
   WHILE (F[1] <= B and F[1] < infinity)
            F[1] := RBFS(N1, F[1], MIN(B, F[2]))
     insert Ni and F[1] in sorted order
RETURN F[1]
```

3.4.2 MA* dan SMA *

MA* dan SMA * adalah algoritma best first search dengan batasan memory yang menggunakan semua memory yang ada.

Algoritma ini menjalankan best first search selama memory masih tersedia, apabila memory penuh maka node dengan nilai terburuk di buang, namun nilainya disimpan pada node atasnya.

3.5 local search

Metode local search bekerja pada formulasi state yang complete. Algoritma ini hanya menggunakan sejumlah kecil node di dalam memory.

Local search berguna untuk menyelesaikan masalah optimasi :

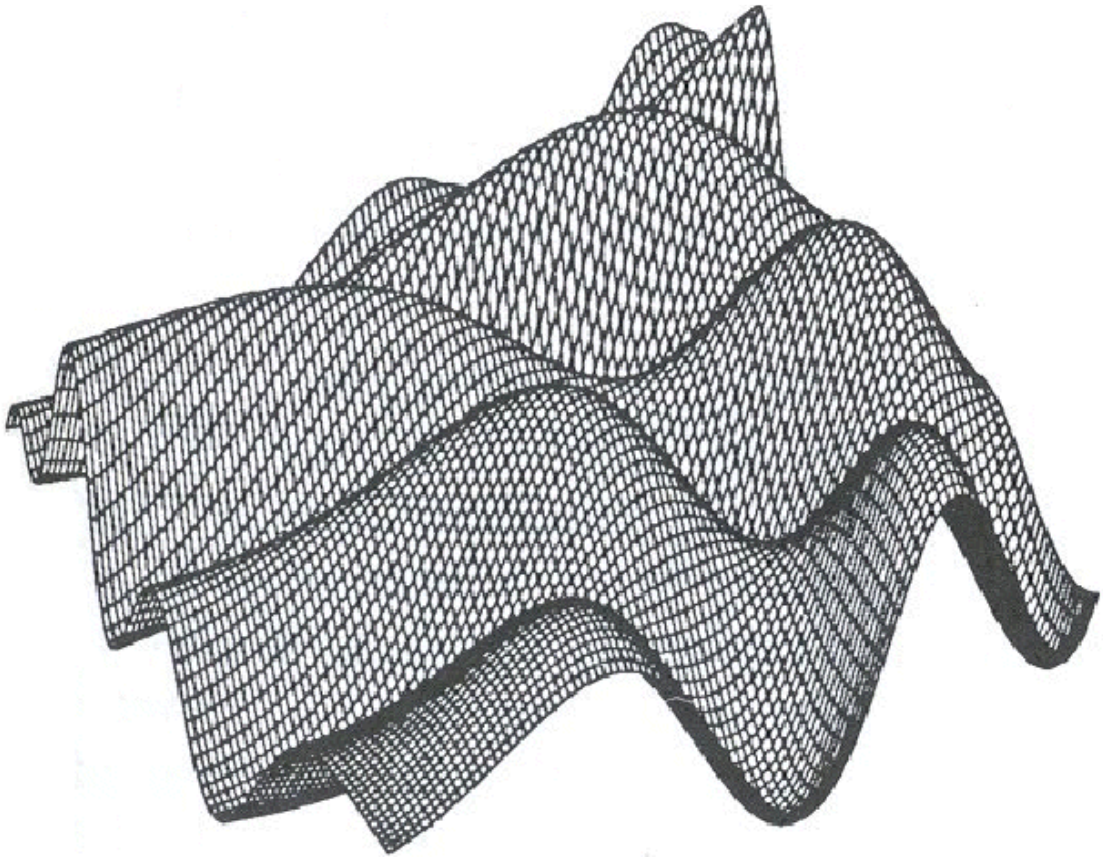
Kadang gampang menemukan solusi

Tapi sulit menemukan solusi terbaik

Tujuan algoritma :

Menemukan konfigurasi optimal

- Hill climbing
- Gradient descent
- Simulated annealing
- Untuk beberapa persoalan deskripsi state mengandung semua informasi yang dibutuhkan, jalur menuju solusi tidak penting
- Contoh :
    - 8 puzzle
    - Map coloring
    - Cyptarithmatic
- Cara kerja : mulailah dari sebuah konfigurasi awal yang melanggar beberapa batasan untuk menjadi sebuah solusi dan mulailah memperbaiki state itu secara bertahap.
- Salah satu cara menggambarkan algoritma perbaikan bertahap (iterative improvement algorithm)_seperti ini adalah dengan membayangkan setiap state yang mungkin dihamparkan pada sebuah lapangan dengan ketinggian setiap state menyatakan baiknya . solusi yang optimal akan mempunyai ketinggian yang paling tinggi. Algoritma iterative seperti ini akan bergerak di lapangan dan mencari puncak dengan hanya melihat sekelilingnya.



3.5.1 itterative improvement

Dalam banyak permasalahan optimasi, jalur menuju goal tidak diperlukan, state tujuan itulah solusinya, contoh dari masalah seperti ini adalah menemukan konfigurasi yang memenuhi batasan tertentu (contoh : 8 queens)

Algoritmanya :

Mulailah dengan sebuah solusi

Perbaiki solusi itu menjadi solusi yang lebih baik

3.5.1.1 contoh

N Queens

Goal : menaruh N buah ratu pada papan n x n dengan tidak ada ratu yang saling menyerang

Contoh :

Konfigurasi ulang papan catur

Disini, state tujuan tidak diketahui namun diketahui di batasi oleh batasan batasan yang harus dipenuhi

Hill climbing (atau gradient ascent/ descent)

Secara bertahap memaksimalkan nilai dari state yang ada sekarang dengan menggantinya dengan state anaknya yang memiliki nilai yang lebih tinggi, selama mungkin.

Catatan : mengecilkan nilai n sama dengan memaksimalkan nilai –n

Karenannya kedua notasi bisa saling menggantikan

Hill Climbing - contoh

Formulasi lengkap state untuk 8 queens

Successor function : pindahkan satu ratu ke kotak lain dalam kolom yang sama

Biaya : jumlah pasangan yang saling menyerang

Masalah minimasi

Algoritma hill climbing :

1. Tentukan successor dari state saat ini
2. Pilih successor yang paling baik (bila successor sama baik pilihlah secara acak)
3. Bila nilai successor lebih kecil dari nilai state saat ini maka berhenti
4. Kalau nilai successor lebih besar maka jadikan successor state saat ini lalu kembali ke langkah 1

- Algoritma ini tidak menyimpan pohon pencarian, hanya menyimpan node saat ini saja
- Algoritma ini beraksi seperti greed y search tapi hanya state yang bisa dicapai dari state saat inilah yang dipertimbangkan
- Algoritma ini menimbulkan beberapa persoalan :

  Local maxima

  Begitu algoritma ini menemukan puncak gunung, ia akan berhenti naik walaupun puncak itu bukan solusi yang terbaik

  Plateaux

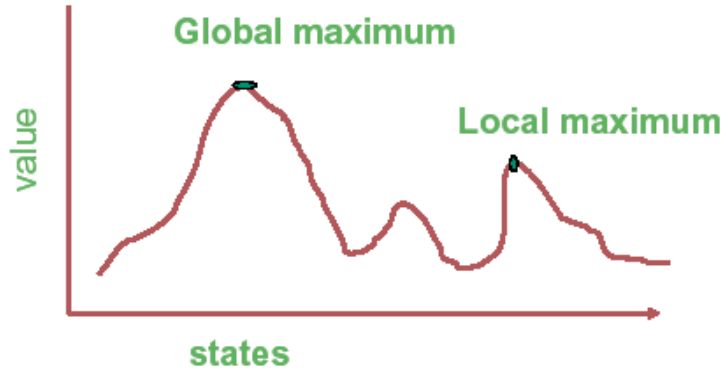
  Bila lapangan rata, berarti semua node memiliki nilai yang sama, maka algoritma akan berjalan secara acak

  Ridges

  Bila algoritma menghadapi masalah yang memiliki punggung bukit maka algoritma akan melakukan jalan zigzag, memperlama proses pencarian

- Bentuk lapangan pencarian sangat mempengaruhi keberhasilan pencarian, sebuah permukaan yang tajam pada kedua sisi dan datar di tengah akan sulit untuk dipecahkan solusinya
- Dapat digabungkan dengan pencarian non deterministik untuk pulih dari local maxima
- Random restart hill climbing adalah sebuah variasi dari algoritma hill climbing dimana ketika setelah mencapai local maxima menyebabkan state saat ini disimpan dan pencarian diulang kembali dari titik awal yang berbeda . setelah beberapa kali berulang, kembalilah ke state yang terbaik yang ditemukan . dengan jumlah ulangan yang cukup metode ini akan menemukan solusi yang optimal.
- Gradient descent adalah versi kebalikan dari hill climbing dimana state yang lebih baik adalah state yang nilai nya lebih kecil. Local minima adalah yang menjadi permasalahan

- *Problem: depending on initial state, may get stuck in local extremum.*

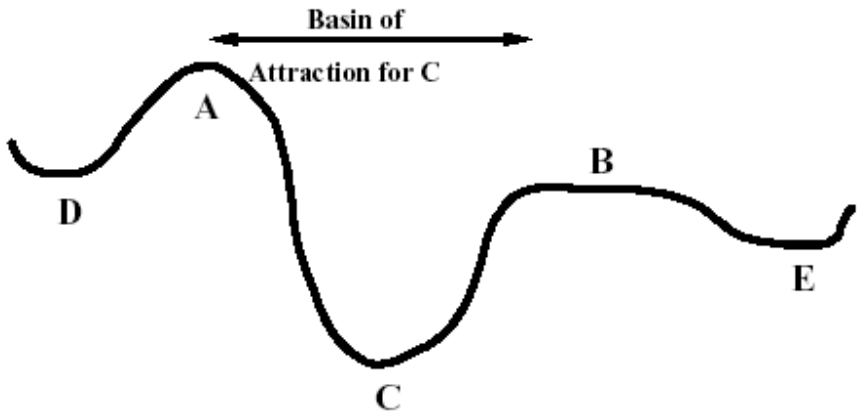


Meminimalisasi energy

Bandingkanlah state space kita dengan keadaan di dunia nyata

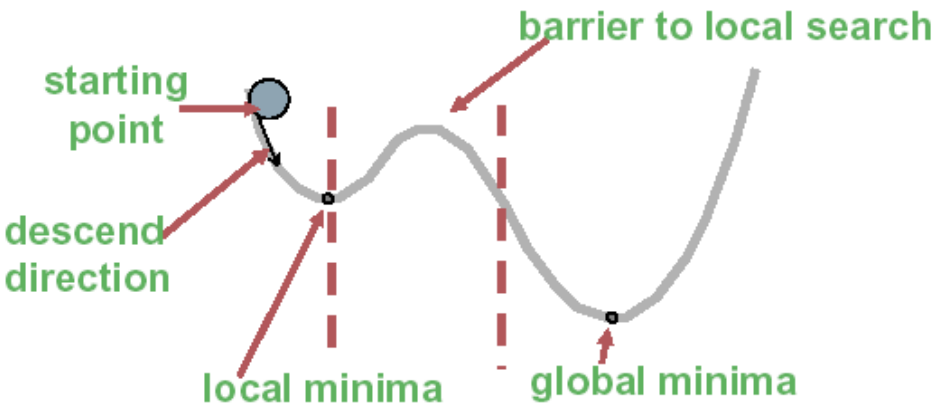
Bandingkan fungsi nilai kita dengan energi potential di alam

Pada setiap perubahan kita dapati DE < 0



Maka dinamika sistem akan menuju ke arah minimum namun ada keadaan minimum bisa terjadi lebih dari sekali.

Pertanyaan : bagaimana cara menghindari local minima ?



Konsekuensi dari kemunculan tanjakan

Simulated annealing : konsep dasar

Dari state awal , pilihlah secara acak satu successor state

- Bila state ini lebih baik dari state awal, terimalah perubahan yang terjadi , jadikan successor state menjadi state saat ini
- Bila tidak lebih baik, jangan dulu menyerah, lemparkan sebuah koin dan terimalah successor menjadi state berikut dengan suatu probabilitas (yang semakin kecil untuk nilai yang lebih tidak baik)
- Jadi kita menerima untuk terkadang tidak mengoptimalisasikan nilai fungsi dengan sebuah probabilitas

Dibandingkan memulai lagi pencarian dengan titik awal yang acak, kita bisa membiarkan algoritma kita mengambil beberapa langkah turun untuk melepaskan diri dari local maxima

Kemungkinan langkah turun ini diatur oleh parameter suhu/ temperature

Suhu tinggi menyatakan tingginya kemungkinan mencoba gerakan yang “tidak baik”, memungkinkan terjadinya pencarian yang non deterministik

Suhu rendah membuat pencarian menjadi lebih deterministik

Suhu awal tinggi dan semakin lama semakin rendah sesuai dengan jadwal annealing yang sudah ditentukan terlebih dahulu

Jadi pada awalnya kita akan mencoba banyak kemungkinan jalan, tapi beriringan dengan waktu secara bertahap akan berhenti di jalur yang paling menjanjikan.

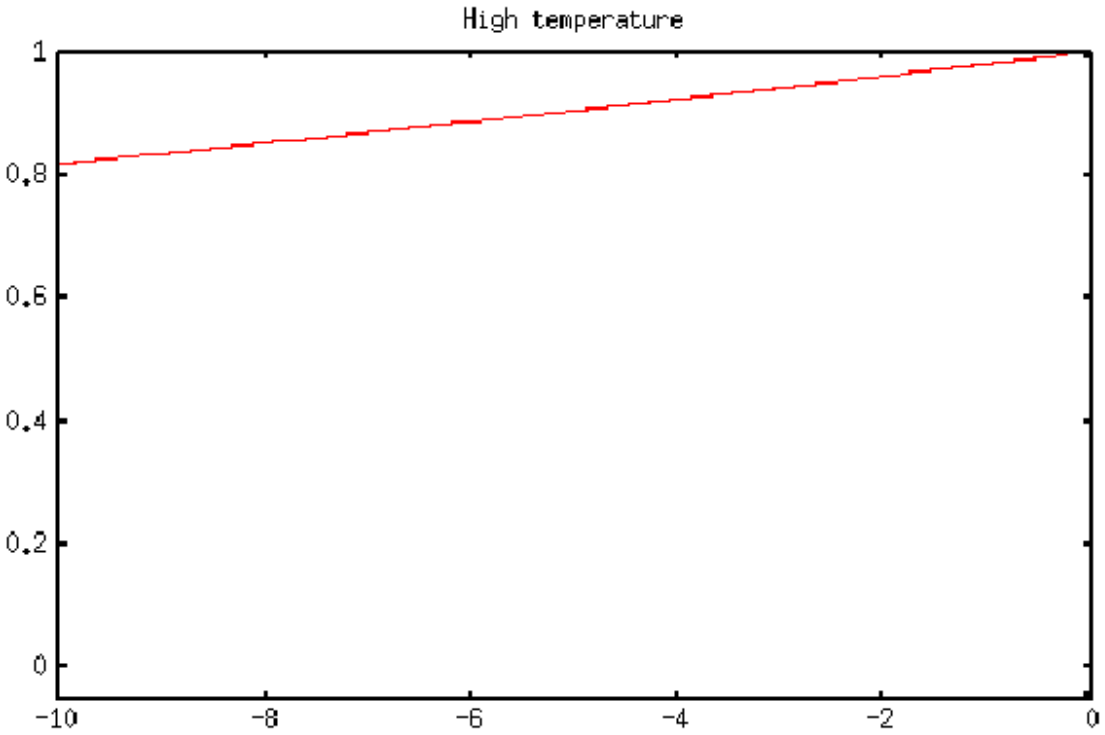
Jika penurunan suhu terjadi cukup lambat maka solusi optimal pasti akan ditemukan.

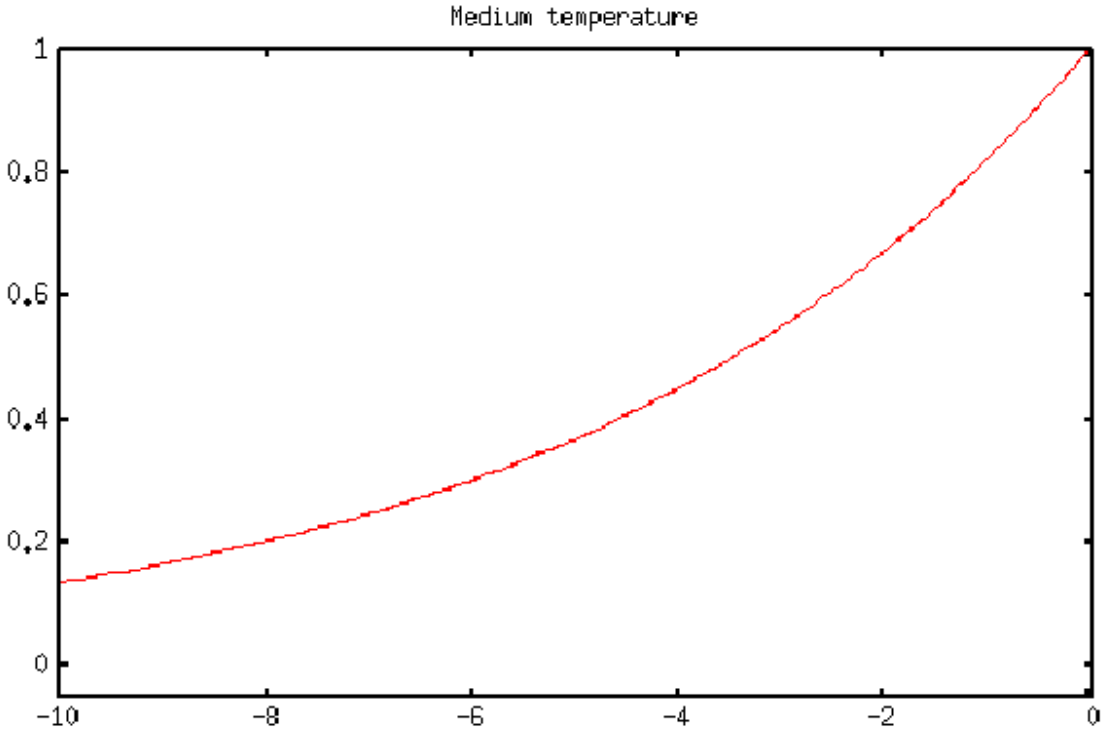
Pada prakteknya : jadwal ini seringkali terlalu lambat sehingga kita cukup puas dengan solusi yang tidak begitu optimal

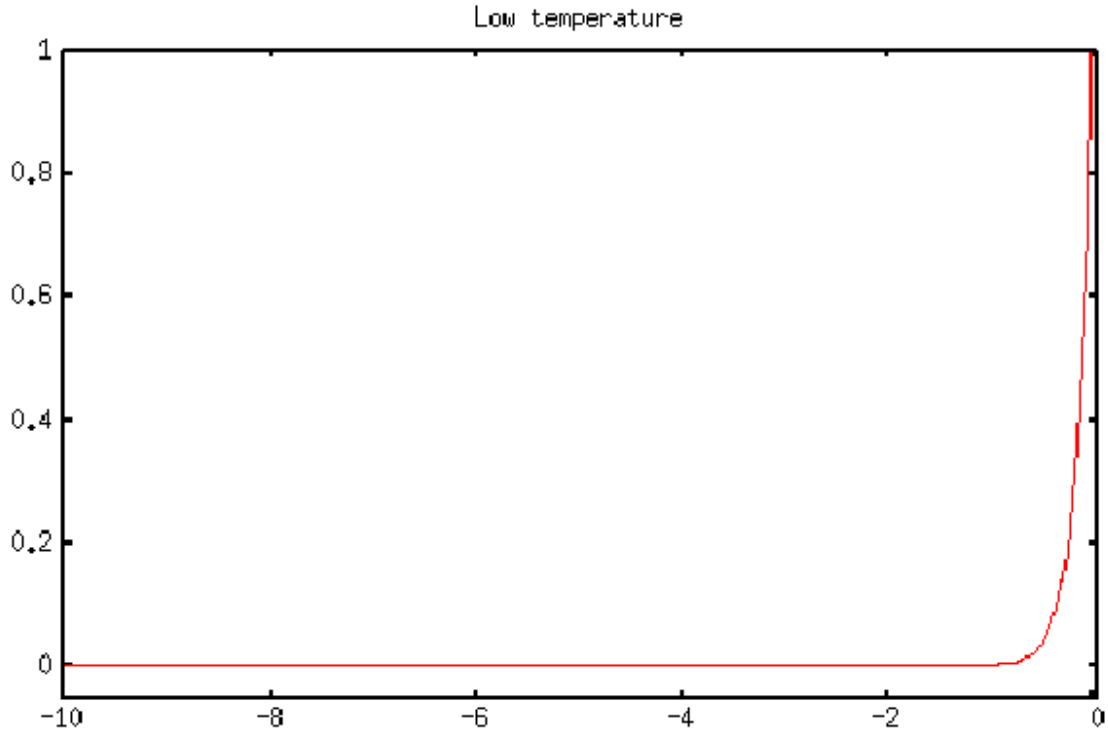
Algoritmanya :

```
set current to start state
for time = 1 to infinity {
   set Temperature to annealing_schedule[time]
   if Temperature = 0 {
      return current
   }
   randomly pick a next state from successors of current
   set ΔE to value(next) - value(current)
   if ΔE > 0 {
      set current to next
   } else {
      set current to next with probability e^(ΔE/Temperature)
   }
}
```

Probabilitas pergerakan turun untuk Delta E negatif pada jangkauan suhu yang berbeda :







Metoda local search yang lain :

Algoritma genetik

Pertanyaan untuk lesson 6

1. Bandingkan IDA * dan A* dalam kompleksitas waktu dan ruang
2. Apakah hill climbing pasti dapat menemukan solusi untuk permasalah N-queen ?
3. Apakah simulated annealing pasti dapat menemukan solusi yang optimum untuk masalah optimasi seperti TSP ?

   Apabila anda memiliki search space sebagai berikut :

State	next	cost
A	B	4
A	C	1
B	D	3
B	E	8
C	C	0
C	D	2
C	F	6
D	C	2
D	E	4
E	G	2
F	G	8

   a. Gambarkanlah state space untuk permasalahan ini
   b. Asumsikan nilai awal adalah A dan state tujuan adalah G, tunjukan bagaimana strategi pencarian di bawah ini akan membuat pohon pencarian untuk menemukan jalur dari awal ke tujuan :
      i. Uniform cost search
      ii. Greedy Search
      iii. A* search

   Pada setiap langkah algoritma, tunjukan node yang dieskpansi, dan isi fringe, laporkan juga solusi yang ditemukan oleh algoritma dan biaya solusi