Tugas Besar I

“N-ything Problem”

IF3170 Inteligensi Buatan

Disusun oleh:

Ranindya Paramitha	**13516006**
Wildan Dicky Alnatara	**13516012**
Rizky Andyno R.	**13516063**
Erma Safira Nurmasyita	**13516072**
Rabbi Fijar Mayoza	**13516081**

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2018

A. Spesifikasi

N-ything problem merupakan modifikasi *N-queen problem*. Perbedaannya, buah catur yang menjadi pertimbangan bukan hanya ratu (*queen*), namun juga meliputi kuda (*knight*), gajah (*bishop*), dan benteng (*rook*). Seperti *N-queen problem*, permasalahan dari *N-ything problem* adalah mencari susunan buah-buah catur pada papan catur berukuran 8x8 dengan jumlah buah catur yang menyerang buah catur lain minimum.

Secara lebih formal, cari susunan buah-buah catur sehingga jumlah pasangan terurut (p, q) di mana p menyerang q minimum. Perhatikan bahwa bila p menyerang q, belum tentu q juga menyerang p. Perhatikan juga bahwa (p, q) dan (q, p) dianggap sebagai dua pasangan yang berbeda.

Untuk menyelesaikan *N-ything problem* ini, kami diminta untuk menggunakan ketiga algoritma *local search* berikut:

1. *Hill climbing*
2. *Simulated annealing*
3. *Genetic algorithm*

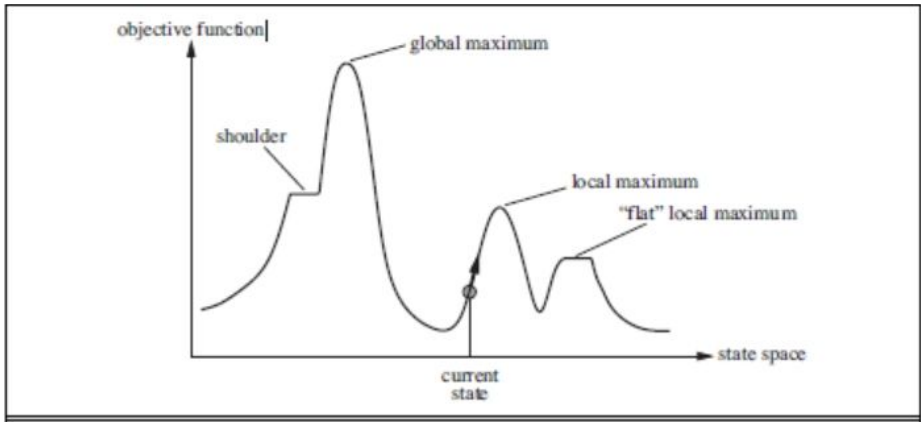
B. Dasar Teori

1. *Local Search*

   Local search adalah algoritma pencarian yang beroperasi dengan cara melakukan penelusuran dari *current state* menuju *next state* yang merupakan *state* tetangganya (*neighbor state*) saja tanpa memerlukan informasi dari keseluruhan *state*. *Local search* hanya menyimpan data dari *current state* dan *next state.* Karena hal itulah, *local search* tepat untuk menyelesaikan permasalahan kontinu yang memiliki jumlah *state* yang tidak terbatas (*infinite*) sehingga tidak menghabiskan banyak memori. Karakteristik permasalahan yang bisa diselesaikan dengan *local search* adalah sebagai berikut.

   - Solusi dari permasalahan tersebut *path-irrelevant*: Solusi akhir tidak memerlukan jalur untuk mencapai *goal* sebagai bagian dari solusi.
   - *Constraint Satisfaction Problem*: Permasalahan yang terpecahkan ketika setiap variabel telah terisi nilai yang tidak menyalahi *constraint* yang telah ditetapkan.

- Permasalahan optimasi: Permasalahan yang memiliki tujuan menemukan *state* dengan nilai terbaik berdasarkan perhitungan fungsi objektif.



Gambar 1. *State-space landscape*

Pada gambar di atas, *goal state* merupakan titik *global maximum. Local search* yang optimal adalah *local search* yang dapat mencapai *global maximum* dan tidak berhenti di titik *local maximum*. Selain itu, *local search* yang baik juga harus *complete* agar jangan sampai solusi tidak ditemukan padahal sebenarnya ada solusi.

2. *Hill Climbing*

Hill Climbing merupakan algoritma *local search* dengan prinsip *greedy* (*greedy local search*). *Hill climbing* dianalogikan seperti seseorang yang menderita amnesia mendaki menuju puncak bukit dalam kabut. Implementasi *hill climbing* merupakan sebuah *looping* yang akan terus menelusuri *neighbor state* apabila nilai dari *neighbor state* lebih tinggi daripada *current state*. *Looping* akan berhenti ketika tidak ada lagi *neighbor state* yang memiliki nilai lebih tinggi. Pencarian akan berhenti dan program mengasumsikan bahwa *goal state* sudah ditemukan karena sudah mencapai puncak. Padahal berdasarkan gambar 1, sebuah puncak belum tentu merupakan puncak tertinggi.

Di sinilah letak kelemahan algoritma *hill climbing.* Algoritma ini tidak mempertimbangkan *state-state* selain *neighbor state.* Hal ini menyebabkan algoritma ini berhenti di *local maximum* dan tidak menyadari bahwa ada sebuah *state* yang merupakan *global maximum,* yang merupakan solusi seharusnya. Oleh karena itu, dapat disimpulkan bahwa algoritma *hill climbing* bisa tidak optimum dan *incomplete,* walaupun tidak selalu

demikian. *Hill climbing* juga tetap memiliki kelebihan, yaitu fakta bahwa algoritma ini memiliki performa yang cepat dalam mencari solusi.

3. *Simulated Annealing*

Simulated Annealing merupakan algoritma perbaikan dari *hill climbing.* Algoritma ini mengkombinasikan *random-walk* dengan *hill climbing. Simulated annealing* memungkinkan agar algoritma bisa keluar dari sebuah *local maximum* (menerima *neighbor state* dengan *score* yang sama/lebih rendah) dengan menggunakan sebuah fungsi probabilitas. Fungsi ini menggunakan parameter yang disebut *temperature*, yang mana parameter ini diisi dengan sebuah nilai, dan seiring berjalannya waktu akan semakin menurun. Menurunnya nilai *temperature*, tergantung pada algoritma penurunan *temperature* yang digunakan (bisa logaritmik, linier, dsb.), *descent rate* (kecepatan penurunan), dan *delay* (penurunan dilakukan setiap berapa langkah sekali).

Jika nilai *temperature* terlalu tinggi, maka *simulated annealing* akan sama seperti *random walk* (semua *neighbor state* baik itu memiliki *score* yang lebih tinggi atau lebih rendah akan diterima). Sedangkan jika terlalu rendah, maka *simulated annealing* akan menjadi *stochastic hill climbing* (hanya *neighbor state* yang lebih baik yang diterima). Nilai parameter *temperature, descent rate,* dan *delay* dapat dikustomisasi untuk mendapatkan solusi secara optimal.

4. *Genetic Algorithm*

Genetic Algorithm (GA) merupakan algoritma *local search* yang menggunakan konsep genetik dan seleksi alam. GA merupakan cabang dari *evolutionary computation*. Pada algoritma ini, dibentuk *population*, yaitu populasi yang merupakan kemungkinan solusi untuk menyelesaikan permasalahan. Populasi ini akan mengalami evolusi dengan mengkombinasikan (*crossover*) individu dalam populasi secara berpasangan untuk membentuk populasi keturunan/ generasi selanjutnya. Dalam melakukan *crossover*, untuk mendapatkan individu yang lebih baik, setiap individu populasi ditentukan *fitness value*-nya terlebih dahulu. *Fitness value* ditentukan berdasarkan nilai objektif kandidat solusi. *Fitness value* ini kemudian digunakan untuk menghitung *survival* (perbandingan *fitness value* suatu

individu terhadap total *fitness value* semua individu dalam populasi) dari masing-masing individu, yang selanjutnya digunakan dalam pemilihan individu yang akan disilangkan. Setiap kali *crossover* dilakukan, individu dalam populasi dapat mengalami mutasi pada elemennya.

C. Implementasi Algoritma

1. *Hill Climbing*

Pada *hill climbing,* pertama-tama akan di-*generate initial state* dengan cara diacak. Kemudian pada langkah selanjutnya program akan berpindah ke *neighbor state. Neighbor state* yang digunakan adalah hasil pemindahan posisi 1 bidak catur dari posisi *current state* menuju posisi yang berbeda. Dari keseluruhan *neighbor state* yang mungkin, dipilih satu *neighbor state* untuk menjadi *state* selanjutnya. *Neighbor state* yang dipilih ini adalah *neighbor state* yang perhitungan fungsi objektifnya menghasilkan nilai heuristik terbaik. Fungsi objektif yang dipakai adalah:

```
f(state) = abs((2*scoreIntersectingDifferentColor) -
scoreIntersectingSameColor)
```

yaitu selisih jumlah konflik antar bidak berbeda warna dengan bidak yang sewarna.

Selanjutnya nilai heuristik *current state* dengan *neighbor state* dibandingkan. Apabila nilai heuristik *neighbor state* lebih baik (lebih tinggi) daripada *current state*, maka *neighbor state* menjadi *current state*. Proses perbandingan ini diulang terus menerus sampai didapatkan *final state* yaitu ketika tidak didapatkan lagi nilai heuristik *neighbor state* yang lebih besar daripada *current state.*

Pseudocode-nya adalah sebagai berikut.

```
current ← MAKE-NODE(problem.INITIAL-STATE)
while TRUE do
      neighbor ← a highest-valued successor of current
      if neighbor.VALUE <= current.VALUE then
            return current
      current ← neighbor
```

2. *Simulated Annealing*

Pada *simulated annealing*, pertama-tama akan di-*generate state* pertama secara acak. Setelah itu, T yang terdefinisi sebagai temperatur akan berkurang seiring waktu setiap langkah dilakukan. Pada setiap langkah, akan dipilih *next state* secara acak dari kumpulan suksesor *current state*. Yang didefinisikan sebagai suksesor *current state* adalah semua *state* yang mana satu bidak dari *current state* berpindah lokasi ke petak yang kosong.

Selanjutnya, ΔE akan dihitung dengan ΔE = nilai *next state* - nilai *current state*. Apabila nilai ΔE lebih dari nol, maka *next state* tersebut diterima dan akan menjadi *current state* pada langkah selanjutnya. Namun, jika ΔE bernilai nol atau kurang dari nol, maka terima *next state* menjadi *current state* dengan nilai probabilitas $e^{\Delta E/T}$.

Pseudocode-nya adalah sebagai berikut:

```
current ← MAKE-NODE(problem.INITIAL-STATE)
while TRUE do
      T ← decreaseTemperature(T)
      if T = 0 then
            return current
      next ← a randomly selected successor of current
      ΔE ← next.VALUE – current.VALUE
      if ΔE > 0 then
            current ← next
      else
            current ← next only with probability e^(ΔE/T)
```

3. *Genetic Algorithm*

Pada *genetic algorithm*, terdapat parameter yang dapat ditentukan oleh pengguna, yaitu:

a. Besar populasi awal yang dibangkitkan (**N**)
b. Banyak generasi yang akan dibangkitkan (**NGen**)
c. Probabilitas *crossover* (**probCross**)
d. Probabilitas mutasi (**probMuta**)

Pada tahap inisiasi, dibangkitkan individu secara *random* sebanyak N yang membentuk suatu populasi. Kemudian dihitung *fitness value* dari tiap individu. *Fitness value* diperoleh dengan menghitung jumlah serangan bidak yang beda warna dikurangi jumlah serangan

bidak sewarna. Setelah itu, dilakukan evolusi sebanyak NGen. Pada tahap evolusi, terdapat dua bagian, yaitu :

a. *Crossover*

Metode penyilangan yang digunakan adalah metode *one point crossover*, yaitu memilih satu titik secara random dan kemudian menggunakan satu titik tersebut sebagai titik penyilangan. Sebagai contoh, bila kita memiliki 2 individu *parent* A dan B yang masing-masing berupa array dengan panjang sama (len) dan titik random yang dibangkitkan adalah x, maka akan menghasilkan 2 *child* C dan D. C memiliki elemen array [0..x-1] = elemen [0..x-1] dari A dan elemen [x..len] = elemen [x..len] dari B. Sedangkan D kebalikannya, memiliki elemen array [0..x-1] = elemen [0..x-1] dari B dan elemen [x..len] = elemen [x..len] dari A.

Pasangan individu yang dipilih sebagai *parent* adalah (1) individu dengan *fitness value* terbaik disilangkan dengan (2) semua individu lain kecuali individu dengan *fitness value* paling buruk. Sebagai contoh, jika ada populasi [A, B, C, D, E] yang mana A merupakan individu dengan *fitness value* terbaik dan E merupakan individu dengan *fitness value* terburuk, maka *parent* yang disilangkan adalah A dengan B, A dengan C, dan A dengan D, sehingga didapat populasi selanjutnya dengan 6 individu.

b. Mutasi

Mutasi yang dilakukan berupa me-*random* ulang posisi sebuah bidak dalam sebuah populasi. Bidak yang dipilih adalah bidak yang paling banyak melakukan serangan ke bidak sewarna.

Crossover dan *mutasi* ini dapat dilakukan dan dapat pula tidak dilakukan bergantung pada probabilitas yang dimasukkan pengguna. Dari populasi yang dihasilkan pada setiap *step* evolusi (*crossover* dan mutasi), dipilih hanya N individu terbaik untuk diproses pada *step* selanjutnya. Jika saat *crossover* dilakukan terdapat 2 bidak yang berada pada lokasi yang sama, maka otomatis akan dilakukan mutasi (perubahan lokasi ke lokasi kosong lain secara acak/*random*) pada bidak yang lebih belakang secara posisi di dalam list. Misalkan bidak pada index 3 dan 7 memiliki lokasi yang sama, maka bidak 7 dimutasi atau dipindahkan ke lokasi lain yang kosong secara acak.

D. Contoh Input-Output

1. Input

   File name: a.txt

   ```
   WHITE QUEEN 2
   WHITE ROOK 2
   WHITE KNIGHT 2
   WHITE BISHOP 2
   BLACK QUEEN 1
   BLACK ROOK 1
   BLACK KNIGHT 1
   BLACK BISHOP 1
   ```

   Contoh 2:

   File name: hill2.txt

   ```
   WHITE QUEEN 3
   WHITE ROOK 2
   WHITE KNIGHT 2
   WHITE BISHOP 2
   BLACK QUEEN 2
   BLACK ROOK 2
   BLACK KNIGHT 1
   BLACK BISHOP 2
   ```

2. *Hill Climbing*

   Contoh 1

   Input file: a.txt

Output:

```
D:\ITB\SEMESTER 5\AI\tubes 1\Nthing-Problem-Solver>py Main.py
>> Please input file name which contains your pawns: a.txt
Loading a.txt . . .
File has been opened successfully.
Which algorithm do you prefer?
1. Hill Climbing
2. Simulated Annealing
3. Genetic Algorithm
>> Input choosen menu (1 or 2 or 3) : 1
..R..r..
....kR..
......KB
...KQ...
.Q...q..
..b.....
...B....
........

3 12
65
```

Contoh 2

Input file: hill2.txt

Output:

```
D:\ITB\SEMESTER 5\AI\tubes 1\Nthing-Problem-Solver>py Main.py
>> Please input file name which contains your pawns: hill2.txt
Loading hill2.txt . . .
File has been opened successfully.
Which algorithm do you prefer?
1. Hill Climbing
2. Simulated Annealing
3. Genetic Algorithm
>> Input choosen menu (1 or 2 or 3) : 1
...kQrR.
.KB.....
..b..Qq.
b.Q.q...
....B.K.
........
...R.r..
........

3 16
97
```

3. *Simulated Annealing*

Contoh 1

Input file: a.txt

Output:

```
D:\ITB\SEMESTER 5\AI\tubes 1\Nthing-Problem-Solver>py Main.py
>> Please input file name which contains your pawns: a.txt
Loading a.txt . . .
File has been opened successfully.
Which algorithm do you prefer?
1. Hill Climbing
2. Simulated Annealing
3. Genetic Algorithm
>> Input choosen menu (1 or 2 or 3) : 2
Do you want to edit algorithm configuration?
1. Yes, I want.
2. No, use default.
>> Input your choice (1 or 2) : 2
...b....
..B.....
.....Q..
...K....
..r.RqRk
.....K..
........
..Q..B..

2 12
49
```

Contoh 2

Input file: hill2.txt

Output:

```
D:\ITB\SEMESTER 5\AI\tubes 1\Nthing-Problem-Solver>py Main.py
>> Please input file name which contains your pawns: hill2.txt
Loading hill2.txt . . .
File has been opened successfully.
Which algorithm do you prefer?
1. Hill Climbing
2. Simulated Annealing
3. Genetic Algorithm
>> Input choosen menu (1 or 2 or 3) : 2
Do you want to edit algorithm configuration?
1. Yes, I want.
2. No, use default.
>> Input your choice (1 or 2) : 2
b.Q.....
..r.....
k..Qr...
..R...K.
....B...
.Q.q....
bK......
.BqR....

7 16
70
```

4. *Genetic Algorithm*

   Contoh 1

   Input file: a.txt

   Output:

```
D:\ITB\SEMESTER 5\AI\tubes 1\Nthing-Problem-Solver>py Main.py
>> Please input file name which contains your pawns: a.txt
Loading a.txt . . .
File has been opened successfully.
Which algorithm do you prefer?
1. Hill Climbing
2. Simulated Annealing
3. Genetic Algorithm
>> Input choosen menu (1 or 2 or 3) : 3
Do you want to edit algorithm configuration?
1. Yes, I want.
2. No, use default.
>> Input your choice (1 or 2) : 2
50-th generation
......Q.
........
...Rq...
....KB..
......k.
.....r.Q
B.b..R..
....K...

0 12
52
```

   Contoh 2

   Input file: hill2.txt

Output:

```
D:\ITB\SEMESTER 5\AI\tubes 1\Nthing-Problem-Solver>py Main.py
>> Please input file name which contains your pawns: hill2.txt
Loading hill2.txt . . .
File has been opened successfully.
Which algorithm do you prefer?
1. Hill Climbing
2. Simulated Annealing
3. Genetic Algorithm
>> Input choosen menu (1 or 2 or 3) : 3
Do you want to edit algorithm configuration?
1. Yes, I want.
2. No, use default.
>> Input your choice (1 or 2) : 2
50-th generation
..Q..q.R
R..q....
...B....
....b...
..r...Qr
b....K..
Q....k..
..B....K

1 16
79
```