LAPORAN
Tugas Besar 1 IF3170 Inteligensi Buatan
"*N-ything Problem*"

disusun oleh:

Rahmat Nur Ibrahim Santosa	13516003
Michelle Eliza Gananjaya	13516015
Ilma Alifia Mahardika	13516036
Muhammad Alif Arifin	13516078

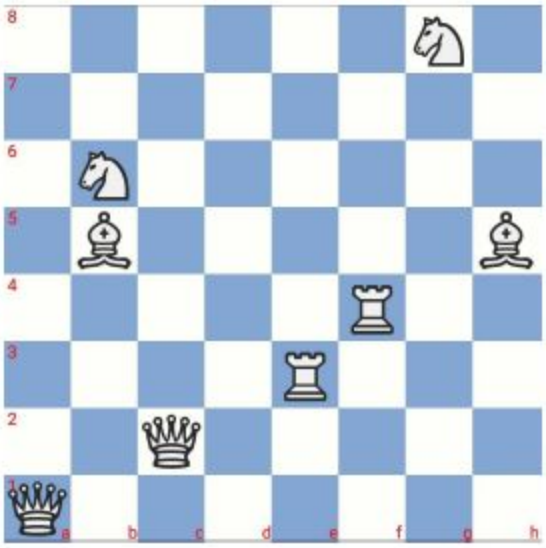
PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2018

A. Deskripsi Persoalan

N-ything problem merupakan modifikasi *N-queen problem*. Perbedaannya terdapat pada buah catur yang digunakan tidak hanya *queen* 'ratu', namun juga meliputi *knight* 'kuda', *bishop* 'gajah', dan *rook* 'benteng'. Seperti *N-queen problem*, persoalan dari *N-ything problem* adalah mencari susunan buah-buah catur pada papan catur berukuran 8x8 dengan jumlah buah catur yang menyerang buah catur lain dengan warna yang sama adalah minimum, namun untuk warna yang berbeda jumlah penyerangan harus maksimum.

Secara lebih formal, mencari susunan bidak catur sehingga jumlah pasangan terurut (p, q) yang artinya p menyerang q minimum. Perlu diperhatikan apabila p menyerang q, belum tentu q juga menyerang p. Perhatikan juga bahwa (p, q) dan (q, p) dianggap sebagai dua pasangan yang berbeda.

Adapun sifat penyerangan ini mengikuti sifat penyerangan pada permainan catur pada umumnya. Misalnya, sebuah benteng dapat menyerang buah catur lain yang berada pada jalur vertikal/horizontal apabila buah catur tersebut tidak terhalang oleh buah catur lainnya, dan seterusnya.



Gambar 1. Contoh solusi N-ything problem

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

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

B. Algoritma

a. *Hill Climbing*

Algoritma *hill climbing* adalah algoritma *local search* yang bersifat iteratif. Algoritma ini dimulai dengan sebuah solusi yang acak, lalu akan dicari solusi yang lebih baik secara inkremental. Algoritma ini akan menemukan optimum lokal dari persoalan, yang belum tentu melupakan optimum global atau solusi yang diinginkan dari persoalan, namun pada banyak kasus hasil dari algoritma ini sudah dapat diterima. Hal ini terjadi karena pencarian ini hanya mengubah satu parameter dan melihat solusi sementara, dan akan terus bergerak hingga tidak ditemukan solusi yang lebih baik dari keadaan sekarang, yang mengakibatkan pencarian dapat terjebak dalam optimum lokal karena sudah dianggap solusi terbaik.

Implementasinya untuk persoalan ini.

i. Hitung *cost* dari kondisi awal papan.
ii. Iterasi semua kemungkinan pemindahan bidak, cari kemungkinan dengan *cost* minimum yang lebih kecil dari *cost* awal. Jika *cost* sama dengan *cost* awal, pilih kemungkinan yang memindahkan bidak yang lebih banyak menyerang bidak lain.
iii. Setelah semua kemungkinan iterasi, jika ditemukan kemungkinan dengan *cost* yang lebih kecil dari *cost* awal, ulangi pencarian.
iv. Jika tidak ditemukan kemungkinan dengan *cost* yang lebih kecil, solusi sudah ditemukan dan pencarian akan dihentikan.

b. *Simulated Annealing*

Algoritma *simulated annealing* adalah algoritma *local search* yang "menggabungkan" *hill climbing* dengan berjalan secara acak dalam beberapa cara yang menghasilkan efisiensi dan ketuntasan. Pada dasarnya algoritma ini mirip seperti proses *annealing* pada metalurgi, yaitu ketika suhu tinggi atom dapat bergerak bebas lalu terjadi pendinginan yang membuat atom yang tadinya bergerak bebas akhirnya menemukan tempat yang optimum. Sama seperti *hill climbing*, *simulated annealing* tidak menjamin akan mendapatkan *global minima/maxima* namun memiliki peluang untuk tidak terjebak pada *local minima/maxima* (karena dapat menerima solusi yang lebih buruk). Hal itu terjadi karena pada dasarnya *simulated annealing* bergerak secara acak dan pada saat temperatur tinggi dapat memilih solusi yang lebih buruk itu dengan harapan tidak terjebak pada *local minima/maxima* namun masih mungkin tetap terjebak pada *local minima/maxima* lainnya. Dan ketika temperatur rendah, *simulated annealing* akan bekerja seperti *stochastic hill climbing*.

Algoritma *simulated annealing* bekerja dengan cara memilih solusi baru secara random lalu akan menerima solusi tersebut apabila lebih baik daripada solusi sebelumnya. Namun apabila lebih buruk, *simulated annealing* dapat menerimanya dengan probabilitas tertentu. Lalu kembali lagi mencari solusi yang baru. Probabilitas menerima solusi yang lebih buruk akan berkurang seiring iterasi. Iterasi berhenti ketika sudah bosan.

Implementasinya untuk persoalan ini.

i. Menentukan temperatur awal (T = 1000)
ii. Mencari solusi sementara (dengan menggerakkan salah satu bidak secara acak, pemilihan bidak juga dilakukan secara acak)
iii. Menghitung *cost* dari solusi sementara. *Cost* dihitung dari jumlah konflik bidak dengan warna yang sama dikurangi dengan jumlah konflik dari bidak dengan warna berbeda.
iv. Mengecek apakah *cost* solusi sementara lebih rendah sama dengan daripada *cost* solusi saat ini.
    1. Apabila lebih rendah sama dengan maka menjadikan solusi sementara menjadi solusi saat ini.
    2. Apabila lebih tinggi maka akan menerima solusi dengan probabilitas yang dihitung dengan menggunakan *boltzman distribution*.
v. Setiap 10 iterasi, temperatur akan menjadi 0,9 * T.
vi. Kembali ke langkah (ii) kecuali masih mendapatkan solusi yang sama selama 500 iterasi berturut-turut (*stuck*, diasumsikan telah mencapai *local/global minima*)

c. *Genetic Algorithm*

Genetic algorithm adalah salah satu variasi dari *stochastic beam search*. Kondisi state solusi didapatkan dari melakukan kombinasi dua *parent states*. Algoritma ini terinspirasi dari *natural selection* pada bidang genetika. *Genetic algorithm* dimulai dengan memproduksi beberapa state random, selanjutnya disebut sebagai populasi. Lalu dalam populasi ini dilakukan penilaian berdasarkan fungsi yang bernama *fitness function*. Semakin tinggi nilai *fitness function* semakin baik state tersebut. Kemudian dilakukan proses *selection*, memilih 'anak' yang paling potensial. Lalu dilakukan proses *crossover*, yaitu menyilangkan dua *parent states* menjadi satu individu. Dan yang terakhir dilakukan *mutation*, yaitu penggantian sebuah nilai dari sebuah individu yang dilakukan secara random. *Genetic algorithm* melakukan eksplorasi secara random.

Implementasinya untuk persoalan ini.

i. Membentuk populasi awal dengan jumlah individu sesuai dengan input dari pengguna
ii. Melakukan pengurutan secara menurun pada populasi tersebut sesuai dengan Fitness Function
iii. Fitness Function dihitung dari jumlah maksimum attack dikurangi cost
iv. Melakukan proses crossover dengan ketentuan:
    1. Individu terbaik 1 dan 2 menjadi parent
    2. Individu terbaik 2 dan 3 menjadi parent
    3. Individu terburuk dibuang
    4. Di akhir crossover dilakukan mutation
v. Ulangi langkah ii sampai mencapai maksimum nilai Fitness Function atau maksimum jumlah iterasi

Karena eksplorasi yang dilakukan *genetic algorithm* sangat random dan dalam program kami terdapat jumlah maksimal langkah, maka mungkin hasil yang didapatkan bukan merupakan global minima.

C. Contoh Input dan Output

a. Menu Utama

```
Please enter filename: test.txt
Choose Local Search Algorithm
1. Hill Climbing
2. Simulated Annealing
3. Genetic Algorithm
Enter the number : 1
```

Khusus untuk *Genetic Algorithm* akan meminta input

```
Please enter filename: test.txt
Choose Local Search Algorithm
1. Hill Climbing
2. Simulated Annealing
3. Genetic Algorithm
Enter the number : 3
Enter number of population: 10
Enter number of maximum iteration: 100
```

b. Hill Climbing

No.	**Input** (test.txt)	**Output**
1	WHITE KNIGHT 2 WHITE BISHOP 2 WHITE ROOK 2 WHITE QUEEN 2 BLACK KNIGHT 0 BLACK BISHOP 0 BLACK ROOK 0 BLACK QUEEN 0	..R..... .R...... .....Q.. ....B .... Q....... ....B.KK 0 0
2	WHITE KNIGHT 2 WHITE BISHOP 2 WHITE ROOK 2 WHITE QUEEN 2 BLACK KNIGHT 2 BLACK BISHOP 2 BLACK ROOK 2 BLACK QUEEN 2	..Kq.bb. ...R.... ..r..QrB ..R..... ....B... ......Q. ....k... ..q...Kk 4 31
3	WHITE KNIGHT 0 WHITE BISHOP 0 WHITE ROOK 0 WHITE QUEEN 8 BLACK KNIGHT 0 BLACK BISHOP 0 BLACK ROOK 0 BLACK QUEEN 0	.Q...... ...Q.... .....Q.. ....Q ....Q... .... Q....... ..Q.Q... 4 0
4	WHITE KNIGHT 0 WHITE BISHOP 0 WHITE ROOK 0 WHITE QUEEN 8 BLACK KNIGHT 0 BLACK BISHOP 0 BLACK ROOK 0 BLACK QUEEN 8	...qQ..q ..q...Q. ...Q..q. .....Q.. ..Qq..Q. ....Q.q. qQ...... ..q..... 6 52

c. ***Simulated Annealing***

No.	**Input** (test.txt)	**Output**
1	WHITE KNIGHT 2 WHITE BISHOP 2 WHITE ROOK 2 WHITE QUEEN 2 BLACK KNIGHT 0 BLACK BISHOP 0 BLACK ROOK 0 BLACK QUEEN 0	..Q..... ....R ...B..K. .B...... ....Q.. K....... ....R... 0 0
2	WHITE KNIGHT 2 WHITE BISHOP 2 WHITE ROOK 2 WHITE QUEEN 2 BLACK KNIGHT 2 BLACK BISHOP 2 BLACK ROOK 2 BLACK QUEEN 2	....Q.q ....k.... q...B.rR B...KbRr ....K ..bQ.k.. 2 44
3	WHITE KNIGHT 0 WHITE BISHOP 0 WHITE ROOK 0 WHITE QUEEN 8 BLACK KNIGHT 0 BLACK BISHOP 0 BLACK ROOK 0 BLACK QUEEN 0	.....Q.. .... .Q...... ....Q... ......Q. ...Q.... Q....... ..Q....Q 2 0
4	WHITE KNIGHT 0 WHITE BISHOP 0 WHITE ROOK 0 WHITE QUEEN 8 BLACK KNIGHT 0 BLACK BISHOP 0 BLACK ROOK 0 BLACK QUEEN 8	...qQ... .... .Q..qQq. .q.Q.... Q..q..Qq ...Q.q.. ..q..Q.. .... 0 72

d. ***Genetic Algorithm***

No.	**Input** (test.txt)	**Output**
1	WHITE KNIGHT 2 WHITE BISHOP 2 WHITE ROOK 2 WHITE QUEEN 2 BLACK KNIGHT 0 BLACK BISHOP 0 BLACK ROOK 0 BLACK QUEEN 0	....R.. ....R... BK...... ....Q ....Q..... B.....K. 0 0
2	WHITE KNIGHT 2 WHITE BISHOP 2 WHITE ROOK 2 WHITE QUEEN 2 BLACK KNIGHT 2 BLACK BISHOP 2 BLACK ROOK 2 BLACK QUEEN 2	....Q.k. ....qK .... b.Qk.... R.r..B.. r.B..K.. ...b.qR. 2 40
3	WHITE KNIGHT 0 WHITE BISHOP 0 WHITE ROOK 0 WHITE QUEEN 8 BLACK KNIGHT 0 BLACK BISHOP 0 BLACK ROOK 0 BLACK QUEEN 0	.......Q ..QQ.... Q....... ......Q. ....Q... .Q...... .....Q.. .... 2 0
4	WHITE KNIGHT 0 WHITE BISHOP 0 WHITE ROOK 0 WHITE QUEEN 8 BLACK KNIGHT 0 BLACK BISHOP 0 BLACK ROOK 0 BLACK QUEEN 8	.... .Q..q.Qq ....Qq.... ....Q.qQ ..q..Q.. ...Q.q.. .q....Qq 2 64