Tugas 1
Light Up Puzzle
Simulated Annealing

Oleh:
Reggie Maurice Gunawan
2016730001

Teknik Informatika
Fakultas Teknologi Informasi dan Sains
Universitas Katolik Parahyangan

Daftar Isi

BAB I

PENDAHULUAN

1.1 Latar Belakang

Light up (Japanese 美術館 *bijutsukan* art gallery), disebut juga Akari, adalah sebuah permainan puzzle logika biner yang dipublikasikan oleh Nikoli. Pada tahun 2011, dia merilis ketiga buku tentang permainan ini beserta aturan-aturan dasarnya.

Pada penelitian ini, penulis akan membuat sebuah algoritma yang akan diimplementasikan pada permainan Light Up. Algoritma yang akan dibuat adalah Simulated Annealing yang merupakan salah satu dari algoritma *local search*.

Hasil yang diharapkan pada penelitian ini yaitu sistem permainan akan dijalankan otomatis dan algoritma dapat menemukan solusi secara mandiri. Hasil keluaran dari algoritma akan bernilai *binary* yaitu **true** atau **false.** Hasil keluaran akan berbeda-beda setiap permainan mulai dijalankan karena posisi papan selalu dibuat secara acak sehingga terkadang algoritma akan menemukan solusi optimal (*global maximum*) ataupun terjebak pada solusi non-optimal (*local maximum*).

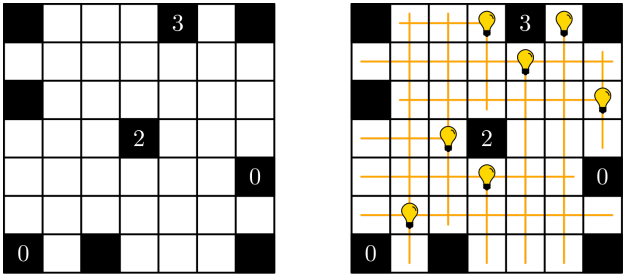
*perihal mengapa permainan ini cocok diterapkan menggunakan *local search* akan dijelaskan pada kesimpulan

BAB II

DASAR TEORI

2.1 Aturan Permainan Light Up

Puzzle dimainkan pada papan berbentuk persegi yang berisi sel hitam dan putih. Pemain menempatkan lampu pada sel putih dan tidak boleh ada lebih dari satu lampu yang saling menyinari, sampai papan terisi cahaya. Sebuah lampu akan mengirim cahaya secara vertikal dan horizontal, menyinari seluruh baris dan kolom kecuali yang ditutupi oleh sel hitam. Sebuah sel hitam bisa memiliki nomor dari rentang nol (0) sampai dengan empat (4), menandakan berapa lampu yang harus diletakkan di sekitar sel hitam tersebut; sebagai contoh, sebuah sel hitam bernilai empat (4) harus memiliki 4 buah lampu yang berada di masing-masing sisinya, dan sebuah sel bernilai nol (0) tidak dapat memiliki lampu di semua sisinya. Sebuah sel hitam yang tidak memiliki nilai dapat memiliki berapapun lampu disekitar sisinya, atau tidak sama sekali. Satu atau lebih lampu yang diletakkan vertikal bertetangga dengan sebuah sel hitam yang memiliki nilai tidak akan dihitung.



Gambar 2.1: Ilustrasi permainan Light Up

2.2 Algoritma Simulated Annealing

2.2.1. Deskripsi

Simulated Annealing (SA) adalah salah satu algoritma untuk optimisasi yang bersifat generik. Berbasiskan probabilitas dan mekanika statistik, algoritma ini dapat digunakan untuk mencari pendekatan terhadap solusi optimum global dari suatu permasalahan. Masalah yang membutuhkan pendekatan SA adalah masalah-masalah optimisasi kombinatorial, dimana ruang pencarian solusi yang ada terlalu besar, sehingga hampir tidak mungkin ditemukan solusi eksak terhadap permasalahan itu. Publikasi tentang pendekatan ini pertama kali dilakukan oleh S. Kirkpatrick, C. D. Gelatt dan M. P. Vecchi, diaplikasikan pada desain optimal hardware komputer, dan juga pada salah satu masalah klasik ilmu komputer yaitu *Traveling Salesman Problem*.

2.2.2. Cara Kerja Algoritma

Simulated Annealing berjalan berdasarkan analogi dengan proses annealing yang telah dijelaskan di atas. Pada awal proses SA, dipilih suatu solusi awal, yang merepresentasikan kondisi materi sebelum proses dimulai. Gerakan bebas dari atom-atom pada materi, direpresentasikan dalam bentuk modifikasi terhadap solusi awal/solusi sementara. Pada awal proses SA, saat parameter suhu (T) diatur tinggi, solusi sementara yang sudah ada diperbolehkan untuk mengalami modifikasi secara bebas.

Kebebasan ini secara relatif diukur berdasarkan nilai fungsi tertentu yang mengevaluasi seberapa optimal solusi sementara yang telah diperoleh. Bila nilai fungsi evaluasi hasil modifikasi ini membaik (dalam masalah optimisasi yang berusaha mencari minimum berarti nilainya lebih kecil/downhill) solusi hasil modifikasi ini akan digunakan

sebagai solusi selanjutnya. Bila nilai fungsi evaluasi hasil modifikasi ini memburuk, pada saat temperatur annealing masih tinggi, solusi yang lebih buruk (uphill) ini masih mungkin diterima, sedangkan pada saat temperatur annealing sudah relatif rendah, solusi hasil modifikasi yang lebih buruk ini mungkin tidak dapat diterima.

Dalam tahapan selanjutnya saat temperatur sedikit demi sedikit dikurangi, maka kemungkinan untuk menerima langkah modifikasi yang tidak memperbaiki nilai fungsi evaluasi semakin berkurang. Sehingga kebebasan untuk memodifikasi solusi semakin menyempit, sampai akhirnya diharapkan dapat diperoleh solusi yang mendekati solusi optimal. Pada temperatur rendah ini, SA biasanya menggunakan konsep Hill-Climbing.

2.2.3. Pemodelan *Pseudo Code*

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  current ← problem.INITIAL
  for t = 1 to ∞ do
      T ← schedule(t)
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← VALUE(current) – VALUE(next)
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(−ΔE/T)
```

Gambar 2.1: *Pseudo Code* Simulated Annealing

BAB III

ANALISIS

2.1 Analisis Pemodelan Algoritma

Berikut merupakan analisis langkah-langkah penyelesaian algoritma Simulated Annealing dalam permasalahan Light Up Puzzle :

1. Inisialisasi map : Membuat pemetaan untuk posisi setiap lampu dan blok beserta nilainya secara acak.
2. Tentukan parameter :
    - α = alpha
    - T = suhu awal
    - Tmin = threshold suhu
    - *Iteration* = jumlah pengulangan iterasi hingga menemukan jawaban
3. Lakukan pengulangan berikut hingga sampai T lebih kecil dari Tmin :
    a) Lakukan perulangan sejumlah *Iteration*
    b) Hitung setiap nilai lampu pada blok
    c) Validasi state pada map
        Jika valid, langsung kembalikan **true**
        Jika tidak, lanjut ke tahap selanjutnya
    d) Turunkan suhu (T)
    e) Kembali ke langkah a
4. Mengembalikan jawaban setelah iterasi selesai.

BAB IV

IMPLEMENTASI

4.1 Bahasa Pemrograman

Pada penelitian ini, bahasa pemrograman yang dipakai adalah Java dengan menggunakan Netbeans sebagai *Integrated Development Environment* (IDE). Penulis memilih Java sebagai platform pengembangan karena dirasa lebih mudah dalam menulis dan memahami cara kerja algoritma dengan konsep *Object Oriented Programming* ketimbang platform lain.

4.2 Implementasi Program

Dalam penerapan pada program, penulis tidak mengutip kode dari sumber ataupun framework apapun karena keterbatasan sumber. Referensi yang dipakai hanya berasal dari PowerPoint materi PSC Complex Search.
Berikut beberapa potongan kode program yang dirasa penulis cukup penting untuk ditunjukkan :

1. Simulated Annealing : Kelas SA

```
public class SA {
    private Map map;
    private double alpha = 0.99;//konstanta penurunan suhu
    private double T = 1;       //suhu awal
    private double TMIN = 0.0001;//threshold suhu
    private int Iteration = 1000;//jumlah iterasi
```

Gambar 4.1: Inisialisasi parameter

Keterangan:

a. Map : inisialisasi papan permainan
b. alpha: konstanta untuk penurunan suhu setiap iterasi
c. T: suhu awal
d. Tmin: batasan minimal suhu sampai mendekati solusi optimal
e. *Iteration*: jumlah iterasi yang akan dilakukan oleh algoritma

Untuk parameter T, Tmin, dan Iteration dapat diubah sesuai kebutuhan. Semakin tinggi jumlah *Iteration,* maka peluang untuk mendapatkan solusi global optimum semakin besar tetapi waktu yang dibutuhkan semakin lama.

```
 Metode Simulated Annealing
*/
public Map generate(){
    Map m = new Map(this.map);
    Map current = new Map(this.map);
    while(T > TMIN){//berhenti jika suhu sudah lebih kecil dari threshold
        for(int i = 0 ; i < Iteration ; i++){
            if(current.calculate() > m.calculate()){//calon jawaban
                m = current;
                if(m.isValid())//memotong jika sudah ada jawaban
                    return m;//mengembalikan jawaban yang valid
            }
            Map n = current.generate();//membentuk solusi baru
            double selisih = n.calculate() - current.calculate();
            double ap = Math.pow(Math.E,(selisih)/T);
            if(ap >= Math.random()){//mengijinkan melakukan kesalahan untuk threshold tertentu
                current = n;
            }
        }
        T *= alpha;//menurunkan suhu
    }
    return m;//mengembalikan jawaban
}
```

Gambar 4.1: Penerapan algoritma Simulated Annealing

Fungsi pada potongan kode tersebut berguna untuk men-generate apakah nilai keseluruhan pada papan permainan sudah mencapai solusi optimal. Lokasi lampu dan blok pada papan permainan dibuat secara acak untuk setiap putarannya (per start).

2. Lokasi lampu (bulb) : Kelas Cell

```
/*
    Kelas ini merupakan Kelas Cell.
    Kelas ini merepresentasikan sebuah titik pada peta Light Up.
 */
public class Cell{
    static final char BLOCK = '-';  // menandakan kalau titik ini merupakan blok bebas
    static final char LIGHT = '+';  // menandakan kalau titik ini dilewati oleh cahaya lampu
    static final char LAMP = 'L';   // menandakan kalau titik ini merupakan sebuah lampu
    static final char SPACE = '_';  // menandakan kalau titik ini kosong (tidak dilewati cahaya)
    static final int INCREASE = 1;  // Kode status yang digunakan untuk menambah nilai cahaya (shine)
    static final int DECREASE = 2;  // Kode status yang digunakan untuk mengurangi nilai cahaya (shine)
    static final int STAY = 3;      // Kode status yang digunakan untuk tidak mengubah nilai cahayay (shine)
    char type = ' ';                // Variable yang menyimpan nilai titik (blok,lampu,cahaya,kosong).
    int shine = 0;                  // Variable yang menyimpan jumlah lampu yang menyinari titik ini.
    int block = -1;                 // Variable yang berguna untuk menandakan blok dengan jumalah lampu disekitarnya
    //nilai block -1 menandakan blok bebas, nilainya akan berganti bila harus terdapat jumlah lampu disekitar blok.
```

Gambar 4.3: Inisialisasi parameter

```
    /*
        Method ini berguna untuk mengetahui apakah titik ini merupakan sebuah blok
        @return boolean menandakan apakah titik ini sebuah blok
     */
    boolean isBlock(){
        return this.type == Cell.BLOCK||((this.type != Cell.LIGHT) && (this.type != Cell.LAMP) && (this.type != Cell.SPACE));
    }
```

Gambar 4.4: Metode untuk mengetahui apakah sebuah sel adalah blok

```
int toggle(){
    if(type == LAMP){
        if(shine > 0){
            shine--;//mengurangi nilai cahaya
            if(shine == 0){
                type = SPACE;//mematikan lampu
            }else{
                type = LIGHT;//titik ini masih di cahayai oleh lampu lain
            }
        }else{
            type = SPACE;//mematikan lampu
        }
        return DECREASE;//mengembalikan kode pengurangan cahaya pada titik lain
    }else if (type == LIGHT || type == SPACE){
        type = LAMP;//titik ini menjadi lampu
        shine++;//nilai cahaya bertambah
        return INCREASE;//mengembalikan kode penambahan cahaya pada titik lain
    }else{
        return STAY;//mengenbalikan kode untuk tidak mengubah nilai cahaya pada titik lain
    }
}
```

Gambar 4.5: Metode *toggle* lampu

Method ini berfungsi untuk mengubah sebuah titik menjadi lampu yang kemudian dicahayai. Metode ini juga dapat mengubah sebuah lampu menjadi space kosong (*toggle off*). Kembalian dari method ini :

- INCREASE: sebuah *int* yang menandakan index (lokasi) tetangga/titik sejalur yang lampunya harus dinyalakan
- DECREASE: sebuah *int* yang menandakan index (lokasi) tetangga/titik sejalur yang lampunya harus dimatikan
- STAY: sebuah *int* yang menandakan tidak mengubah nilai apapun pada titik

```
    /*
        Method ini berguna untuk menambahkan nilai cahaya pada titik ini
        @return boolean menandakan cahaya diteruskan? Atau terhalang oleh blok.
     */
    boolean addShine(){
        this.shine++;
        if(!this.isBlock()){
            if(this.type == SPACE){
                this.type = LIGHT;//mengubah titik ini menjadi cahaya
            }
            return true;
        }else{
            return false;
        }
    }
```

Gambar 4.6: Metode untuk menambahkan cahaya pada titik tertentu

```
/*
    Method ini berguna untuk mengurangi nilai cahaya pada titik ini
    @return boolean menandakan cahaya diteruskan? Atau terhalang oleh blok.
 */
boolean reduceShine(){
    if(shine > 0){
        shine--;
    }
    if(!this.isBlock()){
        if(this.type == LIGHT && shine == 0){
            this.type = SPACE;//mengubah titik ini menjadi kosong
        }
        return true;
    }else{
        return false;
    }
}
```

Gambar 4.7: Metode untuk menghapus cahaya dari titik tertentu

3. Papan permainan: Kelas Map

```
/*
    Kelas ini merepresentasikan state dalam Simulated Annealing
 */
public class Map {

    private Cell [][] map;//atribute peta
```

Gambar 4.8: Inisialisasi parameter

Pada kelas ini, parameter berasal dari Kelas Cell yang merupakan lokasi setiap titik

```
/*
    Method ini berguna untuk mentoggle sebuah titik dan meneruskan pada titik lain.
    @input int row - baris peta
    @input int col - kolom peta
 */
public void toggle(int row, int col){
    int cl = col -1;
    int cr = col+1;
    int ru = row-1;
    int rb = row+1;
    int act = map[row][col].toggle();
    if(act == Cell.INCREASE){//menambahkan cahaya ke 4 arah
        while(cl >= 0 && map[row][cl--].addShine());            //menambahkan cahaya ke kiri titik
        while(cr < map[0].length && map[row][cr++].addShine()); //menambahkan cahaya ke kanan titik
        while(ru >= 0 && map[ru--][col].addShine());            //menambahkan cahaya ke atas titik
        while(rb < map.length && map[rb++][col].addShine());    //menambahkan cahaya ke bawah titik
    }else if(act == Cell.DECREASE){//mengurangi cahaya ke 4 arah
        while(cl >= 0 && map[row][cl--].reduceShine());           //mengurangi cahaya ke kiri titik
        while(cr < map[0].length && map[row][cr++].reduceShine()); //mengurangi cahaya ke kanan titik
        while(ru >= 0 && map[ru--][col].reduceShine());           //mengurangi cahaya ke atas titik
        while(rb < map.length && map[rb++][col].reduceShine());    //mengurangi cahaya ke bawah titik
    }
}
```

Gambar 4.9: Metode *toggle* lampu pada map

Metode ini berfungsi untuk menambahkan/mengurangi cahaya secara bersambungan ke 4 arah baris dan kolom yang dijangkau oleh sebuah titik dengan syarat tidak terhalangi oleh sebuah blok.

```
/*
    Method ini berguna untuk mengembalikan index dari arr
    @return int - index
 */
private int randomBobot(){
    double d = Math.random();
    if(d <= 0.1)return 0;
    if(d <= 0.55)return 1;
    return 2;
}
```

Gambar 4.10: Metode pembobotan posisi

Metode ini berfungsi untuk menghitung bobot dari jumlah titik, blok, dan lampu. Bobot space kosong bernilai 10% dan sisanya untuk lampu dan blok bernilai 55%.

```
/*
    Method ini berguna untuk mengetahui apakah solusi yang dibuat optimal?
    @return boolean - valid
 */
public boolean isValid(){
    for(int i = 0; i < map.length ; i++){
        for(int j = 0 ; j < map[i].length; j++){
            Cell c= map[i][j];
            if(c.type == Cell.SPACE)return false;
            if(c.type == Cell.LAMP && c.shine > 1)return false;
            if(c.isBlock() && c.block>=0){
                int num = 0;
                if(i+1 < map.length && map[i+1][j].type == Cell.LAMP)num++;
                if(i-1 >= 0 && map[i-1][j].type == Cell.LAMP)num++;
                if(j+1 < map[0].length && map[i][j+1].type == Cell.LAMP)num++;
                if(j-1 >= 0 && map[i][j-1].type == Cell.LAMP)num++;
                if(num != c.block)return false;
            }
        }
    }
    return true;
}
```

Gambar 4.11: Metode validasi hasil

Metode ini berfungsi untuk melakukan validasi apakah hasil yang ditemukan sudah merupakan solusi global optimal. Keluaran dari method ini:

- True: nilai keseluruhan lampu pada papan menemukan solusi global optimal
- False: nilai keseluruhan lampu pada papan hanya menemukan solusi lokal optimal (non optimal)

BAB V

PENGUJIAN

Pada bab ini akan dilakukan pengujian dengan memasukkan beberapa parameter berbeda. Parameter yang dimaksud yaitu :

- Alpha : konstanta penurunan suhu
- T : suhu awal
- Tmin : batasan suhu minimal
- *Iteration* : jumlah perulangan yang dilakukan

5.1 Pengujian Parameter Masukan

Pengujian akan dilakukan dengan tes kasus yaitu berupa input parameter yang berbeda-beda. Dilakukan tiga jenis tes kasus, dimana masing-masing tes kasus permainan akan dijalankan sejumlah 5 kali. Diasumsikan bahwa lokasi pemetaan papan permainan tidak diubah.

```
char [][] map = {
        {'_','1','_','0','_','_','_'},
        {'_','_','_','_','_','_','-'},
        {'_','_','2','_','1','_','_'},
        {'-','_','_','-','_','_','-'},
        {'_','_','-','_','2','_','_'},
        {'-','_','_','_','_','_','_'},
        {'_','_','_','1','_','1','_'}
};
```

Gambar 5.1: Kondisi papan permainan

Keterangan:

- _ = space kosong
- - = space yang merupakan blok bebas
- 0 = tidak boleh ada lampu pada titik ini
- 1 = hanya ada 1 lampu pada titik ini, dst (Maksimum 4 lampu pada 1 titik.)

Pada tes kasus yang akan dilakukan, diasumsikan hanya jumlah *Iteration* yang akan diubah karena ingin dibuktikan bahwa semakin besar jumlah iterasi yang dilakukan maka peluang untuk memperoleh solusi optimal semakin besar.

a). Tes Kasus 1

Parameter:

- Alpha = 0.99
- T = 1
- Tmin = 0.0001
- *Iteration* = 500

```
private double alpha = 0.99;     //konstanta penurunan suhu
private double T = 1;            //suhu awal
private double TMIN = 0.0001;    //threshold suhu
private int Iteration = 500;    //jumlah iterasi
```

Gambar 5.2: Parameter masukan tes kasus 1

Hasil :

1. Percobaan ke-1 : false
2. Percobaan ke-2 : false
3. Percobaan ke-3 : true
4. Percobaan ke-4 : false

5. Percobaan ke-5 : false

b). Tes Kasus 2

Parameter:

- Alpha = 0.99
- T = 1
- Tmin = 0.0001
- *Iteration* = 1000

```
private double alpha = 0.99;     //konstanta penurunan suhu
private double T = 1;            //suhu awal
private double TMIN = 0.0001;    //threshold suhu
private int Iteration = 1000;    //jumlah iterasi
```

Gambar 5.3: Parameter masukan tes kasus 1

Hasil :

1. Percobaan ke-1 : false
2. Percobaan ke-2 : true
3. Percobaan ke-3 : false
4. Percobaan ke-4 : true
5. Percobaan ke-5 : false

b). Tes Kasus 3

Parameter:

- Alpha = 0.99
- T = 1
- Tmin = 0.0001
- *Iteration* = 3000

```
private Map map;
private double alpha = 0.99;     //konstanta penurunan suhu
private double T = 1;            //suhu awal
private double TMIN = 0.0001;    //threshold suhu
private int Iteration = 3000;    //jumlah iterasi
```

Gambar 5.4: Parameter masukan tes kasus 2

Hasil :

1. Percobaan ke-1 : false
2. Percobaan ke-2 : true
3. Percobaan ke-3 : true
4. Percobaan ke-4 : true
5. Percobaan ke-5 : true

Berdasarkan tiga pengujian yang sudah dijalankan, dapat dilihat pada hasil percobaan bahwa semakin besar jumlah *Iteration* makan peluang algoritma untuk menemukan solusi optimal semakin besar. Hal tersebut dikarenakan scope yang ditelusuri oleh algoritma akan menjadi lebih besar/luas sehingga dapat meminimalisir terjadinya solusi *local maximum.*

BAB VI

KESIMPULAN

Berdasarkan hasil penelitian permainan puzzle Light Up yang dilakukan dengan menerapkan algoritma Simulated Annealing, maka dapat diambil beberapa kesimpulan sebagai berikut:

1. Permainan puzzle ini cocok diterapkan algoritma *local search* dengan teknik Simulated Annealing karena dengan saking banyaknya kemungkinan untuk setiap titik (lampu on/off), tidak mungkin untuk dilakukan brute force. Selain itu, *local search* pada Simulated Annealing memberikan opsi untuk memasukkan batasan (jumlah iterasi, suhu awal, konstanta penurunan suhu) dalam pencarian agar solusi dapat ditemukan sesuai dengan keinginan kita.

2. Semakin besar jumlah iterasi yang dilakukan maka peluang untuk menemukan solusi optimal (*global maximum*) semakin besar, karena batasan jangkauan pencarian yang semakin luas.

REFERENSI

1. https://en.wikipedia.org/wiki/Light_Up_(puzzle)
2. https://www.geeksforgeeks.org/simulated-annealing/#:~:text=Simulated%20annealing%20is%20based%20on,cools%20into%20a%20pure%20crystal
3. https://www.npmjs.com/package/simulated-annealing
4. http://repository.its.ac.id/63783/1/2510100127-Undergraduate%20Thesis.pdf