# PENCARIAN SHORTEST PATH DINAMIK DENGAN ALGORITMA BELLMAN-BASED FLOOD-FILL DAN IMPLEMENTASINYA PADA ROBOT MICROMOUSE

**Samudra Harapan Bekti – NIM 13508075**

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jalan Ganesa 10 Bandung 40132
e-mail: if18075@students.if.itb.ac.id

## ABSTRAK

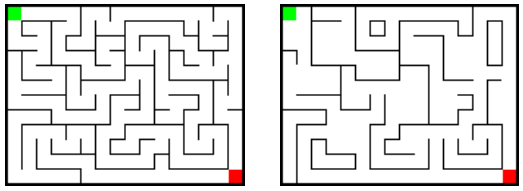
**Makalah ini membahas salah satu metode untuk memecahkan masalah dalam kompetisi robot micromouse. Robot micromouse dituntut untuk dapat mencari jalan menuju suatu tempat tertentu dalam labirin dan kembali ke tempat awal dengan rute seefisien mungkin. Pada keadaan awal, robot hanya mengetahui ukuran labirin tanpa mengetahui pola dinding penghalang yang terdapat dalam labirin. Karena dinding penghalang baru diketahui secara berangsur-angsur selama robot menjelajahi labirin, maka dibutuhkan algoritma pencarian jalur terpendek yang dinamik. Algoritma Flood-Fill merupakan modifikasi dari algoritma Bellman–Ford yang memetakan setiap sel dalam labirin dengan nilai tertentu berdasarkan jaraknya terhadap tempat tujuan. Dari hasil analisis, algoritma ini mampu mencari jalur ke tempat tujuan walaupun labirin yang dihadapi bukan sebuah perfect maze.**

**Kata kunci:** Flood-Fill, labirin, jalur terpendek, micromouse.

## 1. PENDAHULUAN

*Maze solving* merupakan salah satu permasalahan untuk mencari jalan keluar dari suatu labirin. Labirin terbagi menjadi dua menurut susunan dinding penghalang yang terdapat dalam labirin itu sendiri, yakni labirin sempurna (*perfect maze*) dan labirin tidak sempurna (*imperfect maze*). Labirin sempurna tidak memiliki daerah terisolasi, tidak ada jalur sirkuler, dan tidak ada daerah terbuka.

Labirin sempurna dapat dihasilkan dengan modifikasi algoritma Depth-First Search, dan jika labirin yang dihasilkan terhubung sederhana (setiap dinding dalam labirin terhubung dengan dinding terluar labirin) maka untuk mencari solusi dari labirin jenis ini cukup dengan mengikuti dinding kiri atau dinding kanan secara terus menerus.

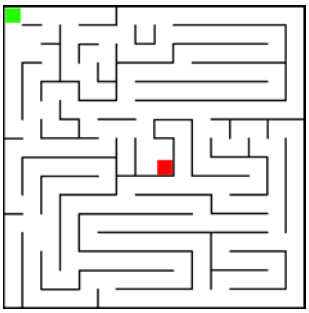


**a** **b**

**Gambar 1.1. (a) Contoh labirin sempurna dan (b) contoh labirin tidak sempurna.**

Kompetisi micromouse sudah dimulai sejak tahun 1970 di berbagai belahan dunia. Micromouse sendiri merupakan suatu robot otomatis yang dilengkapi sensor dan prosesor untuk mencari jalan keluar dari suatu labirin.

Permasalahan mencari jalan keluar dari suatu labirin menjadi menarik apabila labirin yang dihadapi memiliki jalur sirkuler seperti pada kompetisi micromouse di Jepang tahun 1991 berikut:



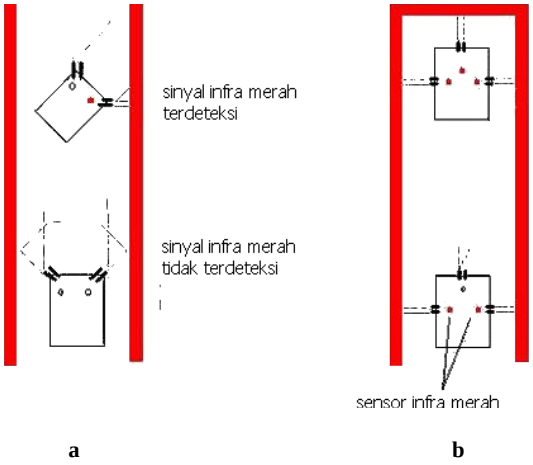
**Gambar 1.2. Labirin kompetisi di Jepang tahun 1991**

Perhatikan apabila robot hanya mengikuti dinding kiri terus menerus maka robot hanya akan kembali ke titik awal dia berada, oleh karena itu dibutuhkan algoritma yang dapat menangani jalur sirkuler seperti ini.

## 2. METODE

Pada dasarnya tujuan kompetisi micromouse adalah mencari jalan menuju sel yang berada di pusat labirin berbentuk bujur sangkar, kemudian kembali ke tempat asal dengan rute yang paling pendek. Hal ini seperti seekor tikus yang mencuri keju yang ada di pusat labirin.

Sebuah robot micromouse memiliki sensor proximity yang dapat mendeteksi dinding dan rintangan lainnya pada labirin. Sensor ini akan mengembalikan nilai yang sebanding dengan jarak robot dengan dinding kiri, dinding kanan, atau dinding di depan robot. Untuk selanjutnya, data dari sensor ini akan disimpan oleh robot untuk pemetaan bentuk labirin.




**Gambar 2. (a) Sensor inklinasi untuk mendeteksi kemiringan orientasi robot dan (b) sensor dinding.**

Algoritma pemecahan labirin telah sejak lama diteliti, tetapi kebanyakan algoritma yang ditemukan tidak cocok untuk robot micromouse karena keterbatasan kecepatan pemrosesan dan ruang penyimpanan data. Beberapa algoritma seperti Wall Folower, Depth-First Search, dan Flood-Fill cukup sederhana sehingga dapat diimplementasikan pada robot micromouse.

### 2.1 Algoritma Wall Follower

Algoritma Wall Follower adalah jenis yang paling sederhana dari teknik *maze solving* lainnya. Prinsipnya adalah mengikuti dinding kanan atau kiri secara terus menerus selama menjelajahi labirin. Ketika sensor pada robot mendeteksi dinding yang terbuka (terdapat jalan) maka robot akan berhenti dan belok pada arah yang bersangkutan, kemudian berjalan lurus dan mengikuti dinding kembali.

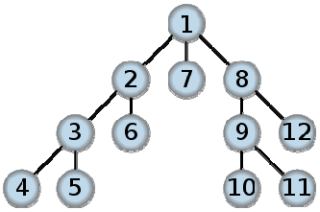
Algoritma ini cenderung berharap menemukan jalan keluar dibanding mencoba menyelesaikan labirin tersebut. Langkah-langkah yang dilakukan adalah sebagai berikut (dalam pseudo-code):

```
If ada jalan ke kanan
    Belok ke kanan
Else If ada jalan ke depan
    Do nothing
Else If ada jalan ke kiri
    Belok ke kiri
Else
    Balik badan
End If

Maju ke depan satu sel
```

Algoritma sederhana seperti mengikuti dinding kanan atau kiri terbukti tidak dapat digunakan pada pola labirin seperti pada Gambar 1.2 yang merupakan sebuah *imperfect maze*. Kebanyakan kompetisi saat ini secara sengaja membuat labirin dengan pola seperti ini, sehingga algoritma ini sudah tidak sesuai untuk digunakan dalam kompetisi micromouse.

### 2.2 Algoritma Depth-First Search

Algoritma Depth-First Search biasa digunakan untuk proses traversal atau pencarian dalam struktur pohon atau graf. Aplikasi algoritma ini pada pohon akan memberikan hasil seperti ini:




**Gambar 2.2. Hasil traversal sebuah pohon dengan algoritma Depth-First Search. Angka disini menunjukkan urutan simpul tersebut dikunjungi.**

Algoritma ini juga dapat diadaptasi menjadi salah satu metode intuitif untuk pencarian dalam labirin. Pertama-tama robot akan berjalan lurus, ketika menemukan percabangan jalan maka robot akan memilih salah satu dari jalan yang tersedia secara acak. Apabila jalan yang dipilih ternyata merupakan jalan buntu, maka robot akan melakukan *backtracking* ke percabangan tadi kemudian memilih jalan yang lain. Proses seperti ini akan membuat robot mencoba semua kemungkinan jalan yang ada. Dengan menjelajahi semua sel dalam labirin, pada akhirnya robot akan dapat menemukan tempat yang dituju.

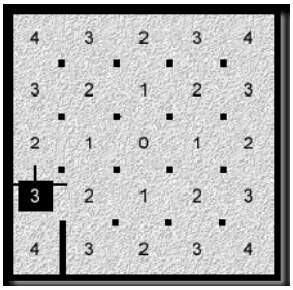
Algoritma ini akan selalu berhasil menyelesaikan labirin ketika pencarian jalur terpendek tidak dibutuhkan, tetapi proses ini akan membutuhkan waktu yang cukup lama tergantung kompleksitas dari labirin yang dihadapi.

## 2.3 Algoritma Flood-Fill

Secara umum algoritma Flood-Fill akan menentukan daerah-daerah yang terhubung dengan suatu simpul dalam array multi-dimensi. Algoritma ini sering digunakan dalam program editor gambar bitmap untuk mewarnai suatu daerah terbatas dengan warna tertentu (*boundary fill*). Algoritma ini dapat diadaptasi untuk menyelesaikan permasalahan labirin secara dinamis.

Algoritma Flood-Fill dapat dianalogikan seperti membanjiri labirin dengan air yang banyak. Air akan terus mengalir hingga mencapai lantai tempat tujuan. Jalur yang dilalui oleh tetesan air pertama di tempat tujuan merupakan jalur terpendek untuk mencapai tempat tujuan tersebut.

Algoritma ini melibatkan proses penomoran setiap sel dalam labirin dimana nomor-nomor ini merepresentasikan jarak setiap sel dengan sel tujuan. Sel tujuan yang ingin dicapai diberi nomor 0 seperti pada Gambar 2.3.1. Apabila robot sedang berada pada suatu sel yang bernomor 1 maka robot tersebut berada sejauh 1 sel dari sel tujuan. Begitu pula apabila robot sedang berada pada suatu sel yang bernomor 3 maka robot tersebut berada sejauh 3 sel dari sel tujuan. Dengan asumsi robot tidak dapat bergerak secara diagonal, nomor-nomor sel pada labirin 5x5 akan berbentuk seperti ini:




**Gambar 2.3.1. Sebuah visualisasi labirin 5x5. Robot sedang berada pada sel yang bernomor 3 dengan orientasi ke arah utara. Perhatikan bahwa sebuah dinding di timur telah terdeteksi pada labirin.**

Ketika robot akan berjalan, robot perlu memeriksa semua sel tetangga yang tidak terhalang oleh dinding, kemudian mencari sel tetangga yang memiliki nomor terkecil. Pada contoh diatas, robot harus mengabaikan semua sel disebelah barat karena terhalang oleh dinding dan robot harus melihat nomor sel yang berada di utara, timur, dan selatan karena sel-sel tersebut tidak terhalang oleh dinding. Sel di utara bernomor 2, sel di timur bernomor 2, sedangkan sel di selatan bernomor 4. Robot perlu menyortir nomor-nomor tersebut sehingga didapatkan sel yang memiliki nomor terkecil. Pada contoh kali ini sel di utara dan timur memiliki nomor yang sama yakni 2. Hal ini berarti robot dapat dengan bebas memilih akan berjalan ke arah utara atau timur—keduanya akan memiliki jarak yang sama menuju sel tujuan. Karena berbelok akan memakan waktu, maka robot lebih baik memilih berjalan ke utara agar dapat langsung bergerak ke sel selanjutnya.

Sejauh ini proses penentuan keputusan dapat dituliskan sebagai berikut (dalam pseudo-code):

```
If sel di utara terhalang dinding
   Abaikan sel di utara
Else
   Push sel di utara ke stack
End If

If sel di timur terhalang dinding
   Abaikan sel di timur
Else
   Push sel di timur ke stack
End If

If sel di selatan terhalang dinding
   Abaikan sel di selatan
Else
   Push sel di selatan ke stack
End If

If sel di barat terhalang dinding
   Abaikan sel di barat
Else
   Push sel di barat ke stack
End If

Pop semua isi stack hingga kosong
Sort isi stack tadi dan pilih sel dengan
     nomor terkecil

Maju ke arah sel dengan nomor terkecil tadi
```

Hingga saat ini algoritma seperti diatas sudah cukup untuk dapat mencapai sel yang dituju tetapi hanya untuk labirin yang tidak memiliki dinding. Karena labirin yang sebenarnya memiliki dinding, maka diperlukan suatu mekanisme pemetaan dinding.

Memori sebesar satu byte cukup untuk menyimpan informasi mengenai dinding. Walaupun satu byte terdiri atas 8 bit, kita hanya membutuhkan 4 bit untuk dijadikan *flag* dinding utara, timur, selatan, dan barat seperti berikut:

```
BIT       7  6  5  4  3  2  1  0
WALL                  U  T  S  B
```

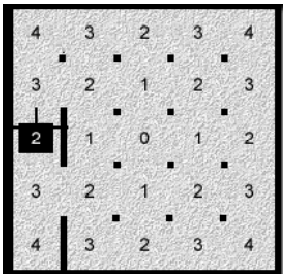
Perlu diingat bahwa setiap sisi dinding dipakai bersama oleh dua buah sel (yang saling bertetangga tetapi terpisah oleh dinding), oleh karena itu proses *update bit-flag* dinding untuk sebuah sel juga dapat dilakukan sekaligus untuk sel tetangga tersebut.

Prosedur untuk meng-*update* bit-flag dinding dapat berbentuk seperti ini (pseudo-code):

```
If sel di utara terhalang oleh dinding
   Pasang bit “utara” pada sel tempat kita
   berdiri
   Pasang bit “selatan” pada sel di utara
   kita
```

```
Else
    Do nothing
End If

If sel di timur terhalang oleh dinding
    Pasang bit “timur” pada sel tempat kita
    berdiri
    Pasang bit “barat” pada sel di timur
    kita
Else
    Do nothing
End If

If sel di selatan terhalang oleh dinding
    Pasang bit “selatan” pada sel tempat
    kita berdiri
    Pasang bit “utara” pada sel di selatan
    kita
Else
    Do nothing
End If

If sel di barat terhalang oleh dinding
    Pasang bit “barat” pada sel tempat kita
    berdiri
    Pasang bit “timur” pada sel di barat
    kita
Else
    Do nothing
End If
```

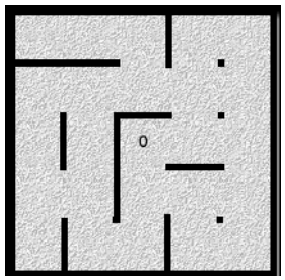
Hingga saat ini telah diperoleh cara untuk memetakan dinding pada labirin. Seiring dengan perjalanan, robot akan menemukan dinding baru, dan diperlukan cara untuk meng-*update* penomoran sel dalam labirin.




**Gambar 2.3.2. Visualisasi masalah sel tetangga dengan nomor terkecil yang terhalang dinding.**

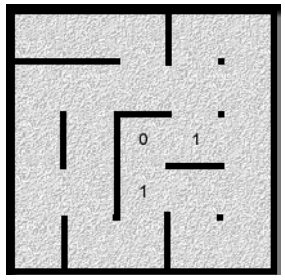
Kembali ke contoh, anggap robot sudah maju satu langkah dan kembali menemukan dinding di sebelah timur seperti pada Gambar 2.3.2. Robot tidak dapat bergerak ke barat maupun ke timur, tetapi bergerak ke utara atau selatan berarti bergerak ke sel yang berjarak lebih jauh karena memiliki nomor yang lebih tinggi. Apabila kondisi seperti ini ditemukan, robot perlu memberi nomor baru untuk setiap sel (*re-flood*) seperti ketika robot pertama kali dijalankan, dengan tetap menyimpan informasi pemetaan dinding sejauh ini.

Sebagai contoh cara memberi nomor pada sel-sel labirin, anggap robot telah cukup lama menjelajahi labirin dan menemukan banyak dinding baru. Prosedur yang dilakukan akan menginisialisasi array multidimensi yang memuat informasi nomor setiap sel, kemudian meng-*assign* nomor 0 pada sel tujuan. Dalam kata lain, tujuan dari robot adalah mencapai sel yang memiliki nomor 0. Dengan cara ini selama proses penyelesaian labirin nomor-nomor pada setiap sel dapat berubah kecuali sel tujuan, yang selalu bernomor 0.



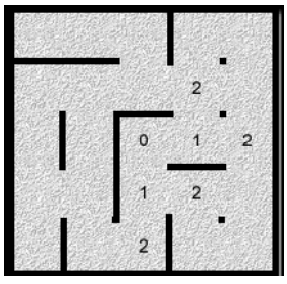

**Gambar 2.3.3. Labirin yang baru diinisialiasi. Sel tujuan yang ditunjukkan dengan nomor 0 terdapat di pusat labirin.**

Prosedur update kemudian akan mencari semua sel tetangga yang terbuka (yakni sel tetangga yang tidak terhalang oleh dinding) kemudian meng-*assign* sel-sel tersebut dengan nomor yang satu lebih tinggi, yakni 1:



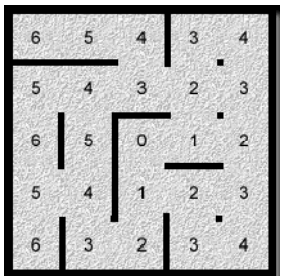

**Gambar 2.3.4. Labirin dengan sel tetangga yang telah diberi nomor satu lebih tinggi dari sel tujuan.**

Prosedur kemudian menemukan kembali sel tetangga yang terbuka dan kembali meng-*assign* sel-sel tersebut dengan nomor yang satu lebih tinggi, yakni 2:




**Gambar 2.3.5. Sel tetangga lain yang telah diberi nomor.**

Prosedur ini diulangi hingga semua sel mendapat nomor baru:




**Gambar 2.3.6. Labirin yang telah selesai diberi nomor.**

Kini dapat diperhatikan cara untuk mencapai sel tujuan dengan jalur terpendek.

Berikut adalah contoh prosedur untuk meng-*update* nomor tiap sel pada labirin (pseudo-code):

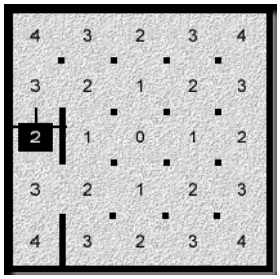
```
Var Level = 0

Inisialiasi array NomorSel sehingga semua
   sel memiliki Nomor = 255
Pasang sel tujuan dalam array bernama
   CurrentLevel
Inisialiasi array baru bernama NextLevel

Begin:
Repeat hingga array CurrentLevel kosong
   Ambil satu sel s dari CurrentLevel
   If NomorSel[s] = 255
        NomorSel[s] = Level
        Tempatkan semua sel tetangga dari s
           ke array NextLevel
   End If
End Repeat

// array CurrentLevel kini telah kosong

If array NextLevel kosong
   // proses penomoran selesai
Else
   Level = Level + 1
   CurrentLevel = NextLevel
   Inisialisasi NextLevel
   Kembali ke Begin
End If
```

Dengan algoritma ini, setiap kali robot mencapai sel baru maka robot perlu menjalankan aksi sebagai berikut:

- *Update* pemetaan dinding
- Isi kembali setiap sel dalam labirin dengan nomor baru
- Tentukan sel tetangga yang memiliki nomor terkecil
- Bergerak menuju sel tetangga yang memiliki nomor terkecil tersebut

## 2.4. Optimasi Algoritma Flood-Fill

Algoritma Flood-Fill yang telah dijelaskan sebelumnya masih dapat dioptimalkan agar mendapatkan hasil yang lebih cepat. Dengan algoritma Flood-Fill yang telah dioptimalkan, robot tidak perlu meng-update nomor setiap sel (*re-flood*) setiap kali diperlukan, melainkan cukup meng-*update* sel-sel yang memang perlu diubah saja. Optimasi ini berlaku untuk kasus seperti pada Gambar 2.3.2 sebelumnya. Untuk kenyamanan, Gambar 2.3.2 telah ditampilkan kembali dibawah ini:

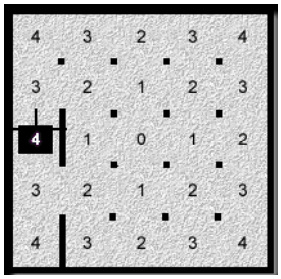



**Gambar 2.3.2. Visualisasi masalah sel tetangga dengan nomor terkecil yang terhalang dinding.**

Pada gambar diatas, Robot tidak dapat bergerak ke barat maupun ke timur, tetapi bergerak ke utara atau selatan berarti bergerak ke sel yang berjarak lebih jauh karena memiliki nomor yang lebih tinggi. Apabila kondisi seperti ini ditemukan, robot perlu memberi nomor baru untuk beberapa sel dengan aturan sebagai berikut:

*Jika sel tempat robot berdiri saat ini bukan merupakan sel tujuan, maka nomor sel ini harus diubah menjadi nomor sel tetangga terkecil yang terbuka (tidak terhalang dinding) ditambah satu.*

Pada contoh diatas, nomor sel tetangga terkecil adalah 3. Menambahkan satu dari nomor ini menghasilkan 3 + 1 = 4. Kini labirin terlihat seperti berikut:




**Gambar 2.4.1. Labirin setelah dilakukan pengubahan terhadap sel tempat robot berdiri.**

Pada beberapa kasus, ketika meng-update nomor sel akan menyebabkan sel tetangga melanggar aturan ”1 + nomor terkecil”. Oleh karena itu perlu diperhatikan agar

hal seperti demikian tidak terjadi. Pada contoh diatas dapat dilihat bahwa sel di utara dan selatan memiliki nomor terkecil 2. Menambahkan 1 pada nomor ini menghasilkan 2 + 1 = 3. Sel di utara dan selatan tidak melanggar aturan, oleh karena itu proses *update* selesai sampai disini dan robot dapat kembali mengikuti jalur seperti biasa. Berikut adalah algoritma untuk mengubah nomor sel (pseudo-code):

```
Buat sebuah stack kosong
Push sel tempat robot berdiri ke stack

Repeat hingga stack kosong
   Pop sebuah sel dari stack
   If nomor sel ini = 1 + nomor terkecil
      dari sel tetangganya yang terbuka
      Do nothing
   Else
      Ubah nomor sel ini menjadi 1 + nomor
      terkecil dari sel tetangganya yang
      terbuka
      Push semua sel tetangganya yang
      terbuka kedalam stack untuk diperiksa
   End If
End Repeat
```

Pada kebanyakan kasus, algoritma Flood-Fill yang dioptimalkan lebih cepat dibanding algoritma Flood-Fill biasa. Dengan meng-update sel yang memang hanya perlu diubah, robot dapat bergerak ke langkah selanjutnya lebih cepat.

## 3. KESIMPULAN

Di dalam makalah ini telah dijelaskan mengenai algoritma Flood-Fill untuk mencari jalan dalam labirin secara dinamis berikut perbandingan dengan algoritma lainnya yang bertujuan serupa. Algoritma Wall Following terbukti gagal dalam menangani *imperfect maze*, sedangkan algoritma Depth-First Search memakan waktu terlalu lama karena mencoba setiap jalan dalam labirin. Algoritma Flood-Fill disini merupakan algoritma yang dapat mencari solusi dengan efisien dengan penggunaan resource yang relatif kecil. Algoritma ini masih dapat dioptimalkan sehingga proses komputasi yang dilakukan dapat diperkecil dengan cukup signifikan, mengingat keterbatasan kemampuan *processing* dalam *embedded devices* seperti robot .

## REFERENSI


[1] Bräunl, Thomas, “Embedded Robotics”, Springer-Verlag Berlin Heidelberg, 2006.

[2] Ananian, C. Scott, “Theseus: A Maze-Solving Robot”, 1997, halaman 10.

[3] De, Tondra, “A detailed design and analysis of Micromouse”, 2004, halaman 40.

[4] http://www.societyofrobots.com/member_tutorials/node/94, “Micromouse – Maze Solver – Theory”, 19 Desember 2009

[5] http://micromouse.cannock.ac.uk/maze/fastfloodsolver.htm, “Faster Flood Solver”, 19 Desember 2009