Metode Heuristik dalam Algoritma Runut Balik

Budiono[1], Adhitya Randy[2], Riza Ramadan[3]

Laboratorium Ilmu dan Rekayasa Komputasi
Departemen Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung

E-mail: if13013@students.if.itb.ac.id[1],
if13027@students.if.itb.ac.id[2], if13037@students.if.itb.ac.id[3]

Abstrak

Permasalahan yang selalu terdapat pada bidang keinformatikaan adalah pencarian metode atau algoritma yang lebih mangkus untuk mencapai solusi. Oleh karena itu dibutuhkan langkah-langkah tertentu yang dapat dijadikan acuan untuk membantu pemecahan masalah tersebut. Misalnya, algoritma runut balik yang sering digunakan untuk program permainan dan kecerdasan buatan. Program permainan sendiri sudah menjadi industri yang sangat besar sekarang ini. Bidang kecerdasan buatan pun sangat penting untuk keamanan jaringan komputer. Algoritma runut balik ini sebenarnya cukup mangkus untuk mencari solusi di antara semua kemungkinan solusi yang ada untuk ukuran persoalan yang kecil. Akan tetapi, waktu komputasi yang dibutuhkan algoritma runut balik biasanya akan meningkat dengan drastis seiring dengan pertambahan ukuran persoalan yang membesar. Karena itulah dibutuhkan metode untuk lebih memangkuskan algoritma tersebut. Salah satu metode yang dapat digunakan adalah menggunakan metode heuristik. Heuristik sendiri memiliki arti, di dalam bidang ilmu komputer, sebagai teknik yang dirancang untuk memecahkan masalah dengan mengabaikan apakah solusi yang dihasilkan dapat dibuktikan (secara matematis) benar [1]. Beberapa metode heuristik dasar yang dapat digunakan antara lain mengurangi kemungkinan solusi, partisi masalah dan penetapan biaya untuk sub-masalah. Walaupu tidak berlaku umum terhadap setiap algoritma, metode heuristik dapat memangkuskan algoritma yang sesuai tanpa mengabaikan solusi.

***Kata kunci**: makalah, karya ilmiah, runut balik, backtrackting, heuristic*

1. Pendahuluan

Runut balik sangat sering digunakan dalam program permainan dan kecerdasan buatan. Dua bidang tersebut sangat erat kaitannya dengan dunia hiburan. Walaupun begitu program permainan dan kecerdasan buatan sudah menjadi komoditas industri permainan di dunia. Selain itu, kecerdasan buatan pun penting untuk keamanan jaringan komputer dan sistem lainnya yang membutuhkan pengambilan keputusan oleh komputer berdasarkan ketentuan yang telah ditetapkan. Metode heuristik yang dapat digunakan untuk mempercepat algoritma runut balik ini pada dasarnya tidak dapat dibuktikan apakah solusi benar secara matematis. Karena itulah tidak ada cara yang pasti untuk mencari metode heuristik ini. Selain itu metode ini tidak berlaku umum untuk tiap permasalahan. Walaupun begitu tetap ada langkah-langkah yang dapat dilakukan untuk mencari metode heuristik ini.

2. Algoritma Runut Balik

2.1. Teori Singkat

Algoritma Runut-Balik tergolong dalam algoritma yang berbasis pada DFS. Algoritma yang berbasis DFS biasanya digunakan untuk mencari solusi persoalan yang lebih mangkus. Algoritma Runut-Balik yang merupakan perbaikan dari algoritma *brute-force*, secara sistematis mencari solusi persoalan diantara semua kemungkinan solusi yang ada. Hanya pencarian yang mengarah ke solusi saja yang selalu dipertimbangkan. Akibatnya, waktu pencarian dapat dihemat. Runut-balik lebih alami dinyatakan dalam algoritma rekursif. Kadang-kadang algoritma ini disebut pula sebagai bentuk tipikal dari algoritma rekursif.

Istilah runut-balik pertama kali diperkenalkan oleh D.H. Lehmar pada tahun 1950. Selanjutnya, R.J. Walker, Golomb, dan Baumert menyajikan uraian umum tentang runut-balik dan penerapannya pada berbagai persoalan. Saat ini algoritma runut-balik banyak diterapkan pada program *games* dan masalah-masalah pada bdang kecerdasan buatan.

2.2. Properti Umum

Persoalan runut-balik dalam konteks algoritma DFS disusun oleh elemen - elemen sebagai berikut :

1. Solusi Persoalan
   Solusi dinyatakan sebagai vektor dengan *n-tupple*.

X = (x1,x2,...,xn), xi E himpunan berhingga Si.
Mungkin saja S1=S2=...=Sn
Contoh : Si = {0,1},
Xi = 0 atau 1

2. Fungsi Pembangkit nilai xk, dinyatakan sebagai
T(k)
T(k) membangkitkan nilai untuk xk, yang merupakan komponen vektor solusi.

3. Fungsi Pembatas(dalam beberapa persoalan fungsi ini dinamakan fungsi kriteria), dinyatakan sebagai
B(x1,x2,.....,xk)
Fungsi pembatas menentukan apakah (x1,x2,....,xk) mengarah ke solusi. Jika ya, maka pembangkitan nilai untuk xk+1 dilanjutkan, tetapi jika tidak , maka (x1,x2,...,xk) dibuang dan tidak akan dipertimbangkan lagi dalam pencarian solusi.

2.3. Prinsip Pencarian Solusi

Langkah – langkah pencarian solusi adalah sebagai berikut :

1. Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan yang dipakai adalah mengikuti metode pencarian dalam (DFS). Simpul – simpul yang sudah dilahirkan dinamakan simpul hidup (*live node*). Simpul hidup yang sedang diperluas dinamakan **simpul-E** (*Expand-node*). Simpul dinomori dari atas ke bawah sesuai dengan urutan kelahirannya.
2. Tiap kali simpul-E diperluas, lintasan yang dibangun olehnya bertambah panjan. Jika lintasan yang sedang dibentuk tidak mengarah ke solusi, maka simpul-E tersebut "dibunuh" sehingga menjadi **simpul mati** (*dead node*). Fungsi yang digunakan untuk membunuh simpul-E adalah dengan menerapkan fungsi pembatas. Simpul yang sudah mati tidak akan pernah diperluas lagi.
3. Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian diteruskan dengan membangkitkan simpul anak yang lainnya. Bila tidak ada lagi simpul anak yang dapat dibangkitkan, maka pencaran solusi dilanjutkan dengan melakukan runut-balik ke simpul hidup terdekat (simpul orangtua). Sealnjutnya simpul ini menjadi simpul E yang baru. Lintasan baru dibangun kembali sampai lintasan tersebut membentuk solusi.
4. Pencarian dihentikan bila telah ditemukan solusi atau tidak ada lagi simpul hidup untuk runut-balik.

3. Metode Heuristik

Pada dasarnya tidak ada langkah-langkah pasti untuk mencari metode heuristik ini. Pencarian ini lebih bersifat intuitif berdasarkan pengamatan terhadap jalannya algoritma terhadap sebuah permasalahan. Walaupun begitu ada tiga metode dasar untuk memnemukan metode heuristik ini. Ketiga metode tersebut adalah mengurangi kemungkinan solusi, partisi masalah dan penetapan biaya untuk sub-masalah.

3.1. Pengurangan Kemungkinan Solusi

Ide dasarnya adalah menghilangkan secara drastis sejumlah kemungkinan solusi. Salah satu pendekatan yang dapat dilakukan adalah memperketat kriteria yang harus dipenuhi di dalam fungsi kriteria. Bahkan dapat ditambahkan fungsi kriteria lainnya untuk pengecekan kemungkinan solusi [3].

3.2. Partisi Masalah

Metode ini selalu berusaha untuk membagi-bagi setiap permasalahan yang ada ke dalam upa masalah yang diasumsikan lebih mudah untuk diselesaikan. Metode ini mempunyai cara berpikir yang mirip dengan algoritma *Divide and Conquer*. Proses partisi ini dapat dilakukan berdasarkan prioritas masalah atau urutan waktu masalah [3].

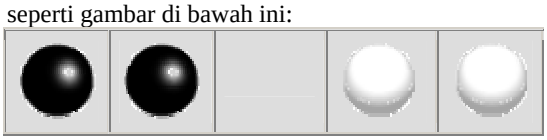
3.3. Penetapan Biaya untuk Sub-Masalah

Metode ini akan menambahkan properti biaya ke dalam setiap kemungkinan solusi. Setiap pengambilan keputusan berdasar pada fungsi kriteria dan juga properti biaya ini yang selalu dihitung berdasarkan langkah sebelumnya. Biaya yang diambil adalah biaya yang terkecil, dengan harapan solusi akan tercapai dengan biaya yang sekecil-kecilnya [2].

4. Contoh Kasus dan Analisis (*Cindy's Puzzle*)

4.1. Permasalahan *Cindy's Puzzle*

Pemain memiliki *n* buah gundu berwarna hitam dan *n* buah gundu berwarna putih, dan memiliki sebuah papan permainan yang berbentuk lajur dan terdiri dari 2*n* + 1 ruang untuk meletakkan gundu di dalamnya. Tempatkan semua gundu hitam pada satu sisi (bagian kiri), semua gundu putih pada bagian yang lain (bagian kanan), dan satu buah ruang kosong di antara kedua sisi tsb. Misalnya susunan awal untuk 4 buah gundu (2 hitam dan 2 putih) seperti gambar di bawah ini:



Tujuan permainan ini adalah membalikkan posisi gundu menjadi seperti ini:

Aturan yang digunakan adalah: gundu hitam hanya dapat dipindahkan ke kanan, dan gundu putih hanya dapat dipindahkan ke kiri (tidak boleh berbalik). Pada setiap perpindahan, sebuah gundu dapat:

1. Maju satu posisi jika ruang di depannya kosong, atau
2. Melompati tepat satu buah gundu yang berbeda warna jika ruang di depan gundu yang dilompati tersebut kosong.

4.2. Algoritma Runut Balik

Algoritma runut balik disusun dalam bentuk prosedur rekursif dangan prioritas gerak LompatKeKiri, GeserKeKiri, LompatKeKanan, dan GeserKeKanan. Dalam notasi pseudo-code, realisasi algoritma runut balik adalah:

```
Procedure puzzle(input: dari, ke : integer)
{ArrGundu adalah array of integer yang
menunjukkan posisi gundu}
Algoritma
If (not solusi)
Then
        LompatKeKiri(); puzzle(dari, ke);
        GeserKeKiri(); puzzle(dari, ke);
        LompatKeKanan(); puzzle(dari, ke);
        GeserKeKanan(); puzzle(dari, ke);
        If (not solusi);
        Then backtrack(ke, dari);
```

4.3. Algoritma Runut Balik dengan Metode Heuristik

Berdasarkan pengamatan pada saat berjalannya algoritma maka dapat terlihat adanya kecenderungan dalam masalah ini. Kecenderungan itu adalah sesudah sebuah gundu bergerak ke kanan dan langkah sesudahnya gundu lain bergerak ke kiri maka akan dilakukan backtrack (tidak menuju solusi). Berdasarkan pengamatan ini, partisi masalah dengan mengelompokkan LompatKeKiri dan GeserKeKiri menjadi GerakKeKiri, kemudian LompatKeKanan dan GeserKeKanan menjadi GerakKeKanan.

Kemudian tambahkan kriteria prioritas gerak. Pada awalnya prioritas gerak adalah sebagai berikut LompatKeKiri, GeserKeKiri, LompatKeKanan, dan GeserKeKanan. Tetapi jika ada gundu yang GerakKeKanan maka prioritas gerak menjadi LompatKeKanan, GeserKeKanan LompatKeKiri, dan GeserKeKiri. Kemudian jika ada gundu yang GerakKeKiri maka ubah prioritas gerak ke keadaan awal. Dalam notasi pseudo-code, realisasi algoritma ini adalah sebagai berikut:

```
Procedure puzzle(input: dari, ke : integer)
{ArrGundu adalah array of integer yang
menunjukkan posisi gundu}
Algoritma
If (not solusi)
Then
        LompatKeKiri(); puzzle(dari, ke);
        GeserKeKiri(); puzzle(dari, ke);
        LompatKeKanan(); puzzle2(dari, ke);
        GeserKeKanan(); puzzle2(dari, ke);
        If (not solusi);
        Then backtrack(ke, dari);

Procedure puzzle2(input: dari, ke : integer)
{ArrGundu adalah array of integer yang
menunjukkan posisi gundu}
Algoritma
If (not solusi)
Then
        LompatKeKanan(); puzzle2(dari, ke);
        GeserKeKanan(); puzzle2(dari, ke);
        LompatKeKiri(); puzzle(dari, ke);
        GeserKeKiri(); puzzle(dari, ke);
        If (not solusi);
        Then backtrack(ke, dari);
```

4.4. Perbandingan Hasil

n	Tanpa Heuristik (jumlah gerak)	Dengan Heuristik (jumlah gerak)
2	20	10
5	351	55
10	14.440	210
15	443.193	465
20	13.467.250	820
25	415.695.771	1275

Dari tabel dapat terlihat bahwa dengan menggunakan metode heuristik, jumlah gerak dapat berkurang drastis, yang berakibat waktu komputasi pun lebih cepat.

5. Kesimpulan

Metode heuristik tidak dapat dibuktikan apakah solusi benar secara matematis. Tetapi algoritma apapun dapat menjadi lebih mangkus dengan menerapkan metode heuristik yang sesuai tanpa mengorbankan solusi. Metode heuristik tidak berlaku umum pada setiap algoritma.

Daftar Pustaka

[1] Rinaldi Munir. 2005. "*Strategi Algoritmik*". Laboratorium Ilmu dan Rekayasa Komputasi – Institut Teknologi Bandung.

[2] **http://www.answers.com** . Diakses tanggal 17 Mei 2005 pukul 20.00

[3] EA Silver. 2004. "An Overview of Heuristic Solution Method". The University of Calgary.