Computer Chess Dengan Algoritma MinMax dan Alpha-Beta

Adityo August P

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung

Kampus ITB Jl. Ganesha no.10 Bandung

e-mail : august_22_raitei@yahoo.com/if16051.students.if.itb.ac.id

ABSTRAK

Dalam kehidupan kita sehari-hari, permainan sudah menjadi suatu kebutuhan. Selain untuk mengisi waktu luang dan melepas jenuh, permainan juga berperan dalam membangun kreatifitas sesorang. Salah satu permainan yang paling menuntut kreatifitas adalah catur. Catur bahkan sudah menjadi cabang olahraga yang dilombakan dalam skala internasional. Catur juga sudah banyak dikembangkan di komputer dengan disertai suatu intelegensi buatan sebagai lawan main. Dalam makalah ini, saya ingin memperlihatkan dua pendekatan intelegensi buatan catur dengan beberapa pendekatan algoritma yang umum digunakan untuk memcahkan solusi ini.

Kata kunci : *Computer Chess, Min Max, NegaMax, Alpha-Beta*

1. Pendahuluan

Permainan catur merupakan suatu cabang olah raga yang sangat menuntut pemikiran kreatif dan pemikiran taktis dari pemainnya. Suatu pemain catur akan diajak untuk bermain dengan 16 bidak catur yang mempunyai aturan gerakan masing-masing.

Permainan catur merupakan permainan dengan dua orang pemain. Tujuan utama seorang pemain catur adalah untuk dapat melakukan *checkmate* (keadaan dimana raja lawan tidak dapat melakukan gerakan lagi). Cara yang diperlukan untuk mendapatkan kemenangan ini adalah dengan mengungguli langkah pemikiran dari lawan.

Pembuatan makalah ini ditujukan untuk menjabarkan pemikiran lawan bermain catur berupa suatu algoritma. Dalam makalah ini akan dibandingkan tiga buah algoritma untuk memainkan permainan catur. Pembahasan

2. Computer Chess

Permainan catur berumur sangat tua. Permainan ini di desain oleh seorang penemu Hungaria bernama Baron Wolfgang von Kempelen pada tahun 1769. dengan permainan penemuannya tersebut ia telah mengalahkan banyak pemain dan termasuk di dalamnya Napoleon Bonaparte.

Dan dalam perkembangan teknologi sekarang sudah sangat banyak permainan catur digital. Sedangkan permainan catur itu sendiri merupakan permainan yang melibatkan dua pemain. Sehingga dalam catur digital, pemrograman suatu pemain maya dengan tingkat kesulitan tertentu menjadi suatu masalah kompleks yang harus dipecahkan.

Gambar 2.1 : *Chess Computer Chessmaster* edisi 10 di Windows XP

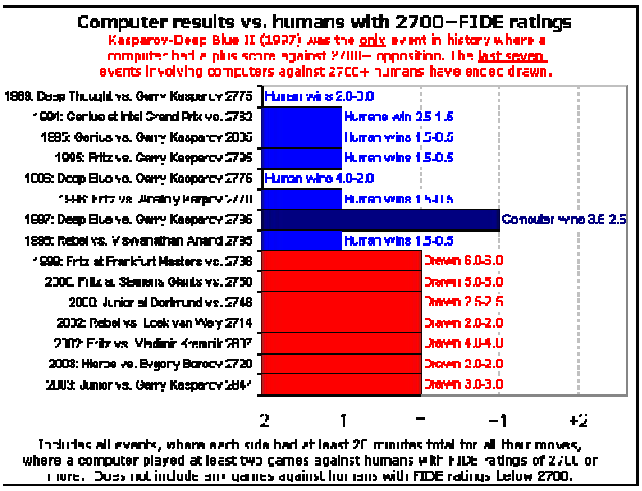
Program catur yang pertama dibuat oleh seorang penemu Jerman bernama Konrad Zuse pada tahun 1942-1945 dengan suatu bahasa pemrograman level tinggi saat itu Plankalkul.

Kemudian pada tahun 1950 seorang matematikawan Amerika, Claude Shannon menulis dua buah artikel yang berisi bagaimana cara memrogram komputer untuk bermain catur. Beliau mengemukakan dua buah tipe pengembangan yang dikenal dengan *Type A* dengan menggunakan algoritma *Brute Force* dan *Type B* dengan algoritma min max.

Suatu perbedaan mendasar antara seorang pemain catur yang sudah mahir dengan seorang pemain baru adalah dalam pengalamannya. Seiring dengan bertambahnya pengalaman, kemampuan untuk *pattern recognizing* suatu kondisi menjadi lebih akurat.

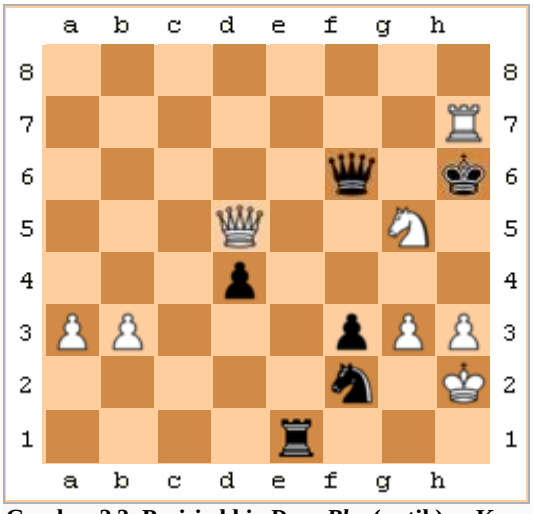
Dalam pemrograman catur, hal *pattern recognizing* inilah yang akan menjadi suatu masalah utama karena untuk manusia, *pattern recognizing* dapat dilakukan tanpa suatu perhitungan dan hasil yang didapat pun sudah sangat dalam dan cukup akurat. Sehingga manusia dapat menyikapi kondisi dia dengan suatu langkah yang dia rasa paling optimal.

Sedangkan untuk komputer, setiap langkah merupakan suatu *patern recognizing* yang harus diperhitungkan terus menerus untuk mengambil suatu langkah optimum.



Gambar 2.2: Hasil statistik pertandingan manusia vs komputer

Puncak dari pertandingan antara manusia dan komputer adalah pada tahun 1997. Pada saat itu juara dunia catur pada masa itu, Gary Kasparov dikalahkan oleh sebuah *computer chess* yang bernama *Deep Blue*. *Deep Blue* merupakan salah satu produk superkomputer IBM yang dibangun berdasarkan bahasa C dan berjalan dalam AIX O/S. *hardware* dari *Deep Blue* memang sengaja dirancang untuk *computer chess*. Dengan arsitekutr berupa *node* 30 paralel, dan VLSI *chess chip*, dan *hardware* lainnya menjadikannya mampu menganalisa 200 juta kemungkinan posisi per detik dua kali lebih cepat dari versi sebelumnya di tahun 1996. Untuk pengembangan *chess computer* lainnya dibentuk Fritz dan generasi-generasi penerus lainnya.



Gambar 2.3: Posisi akhir *Deep Blue*(putih) vs Kasparov(hitam) game I.

3. Algoritma *Computer Chess*

Dalam algoritma *computer chess*, semua langkah yang diambil pemain dimisalkan sebagai sebuah *node* sehingga membentuk suatu *n-ary* pohon raksasa. Dengan mengambil permisalan bahwa rata-rata sebuah *node* akan mempunyai 36 buah percabangan ditambah lagi dengan kemungkinan sudah memperhitungkan semua peraturan legal permainan catur, maka diperkirakan akan terdapat 10^{128} posisi bidak catur yang meungkin terbentuk. Sedangkan di alam semesta ini saja baru dimukan 10^{80} atom.

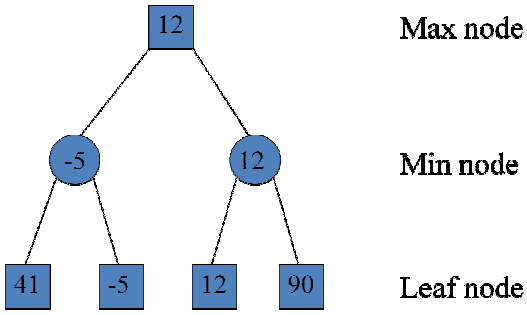
Dengan sebuah permisalan lagi bahwa dengan sebuah supercomputer dengan kecepatan memproses *node* sebesar 100 juta *node* per detik, maka untuk bias menyelesaikan suatu permainan catur yang terdiri atas 40 langkah legal akan diperlukan 10^{113} tahun. Hal ini tentunya merupakan suatu perhitungan dengan algoritma *brute force* yang didefinisikan oleh Shannon sebagai pemecahan solusi *Type A* sangat tidak disarankan. Sehingga untuk memperingkas kompleksitasnya, Shannon mengemukakan pemecahan *Type B* yang bernama algoritma *MinMax*.

3.1 Algoritma *Min Max*

MinMax sendiri sebenarnya merupakan suatu pengembangan sederhana dari algoritma BFS. Kita memisalkan suatu posisi sebagai akarnya dan mulai menyusun anak-anaknya yang merupakan sebuah langkah legal dalam permainan catur. Untuk penelurusan kedalamannya, merupakan suatu perjanjian untuk perkiraan berapa langkah yang akan dihitung. Kemudian *MinMax* akan dihentikan ketika sudah mencapai kedalaman tersebut dan kemudian mulai menelusuri *node* yang belum sampai kondisi berhenti.

Pengembangan algoritma BFS yang diimplementasikan di *MinMax* adalah terdapat dua buah jens *node* yaitu *Max Node* dan *Min Node*.

Kedua *node* ini pada dasarnya adalah sama dalam hal representasinya, sama-sama merepresentasikan posisi langkah yang mungkin. Perbedaannya terletak pada pengisian atau penggabungan *node* yang berasal dari *node* anaknya. *Max node* selalu membandingkan semua nilai yang dimiliki anaknya dan memilih nilai terbesar yang dimiliki anaknya. *Min node* mengecek semua nilai anaknya dan memilih nilai terendah yang dimiliki simpul anak-anaknya.

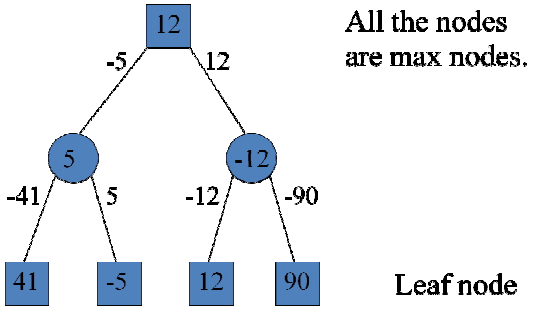


Gambar : Ilustrasi algoritma *MinMax*

Namun algoritma ini juga menemui masalah dalam aplikasi di dunia *computer chess*. Masalah yang kemudian ditemukan adalah kesulitan dalam penandaan *node*. *Node* yang terbentuk selain *node* daun merupakan campuran dari *node max* atau *min.* Sehingga jika dalam *computer chess* diinginkan pengecekan 4 langkah ke depan, akan terdapat suatu algoritma pencocokan tambahan di setiap level langkah. Algoritma itu ditujukan untuk identifikasi apakah *node* level tersebut adalah *node max* atau *node min.*

Untuk pemecahan masalah ini, dikembangkan suatu algoritma turunan *MinMax* yang bernama

algoritma *NegaMax*. Untuk konsep representasinya sudah sama persis dengan algoritma *MinMax*. Namun suatu pembeda algoritma ini adalah kemampuannya untuk men-*generate node* yang akan dibentuk di level atasnya menjadi negatif.



Gambar : Ilustrasi algoritma *NegaMax*

Dengan mengubah semua nilai *node* yang akan dibandingkan menjadi negatif, kita hanya perlu mengambil nilai maksimum dari semua *node* anak dan diisikan ke *node* bapaknya. Sehingga kita dapat memangkas pengecekan apakah *node max* atau *node min* yang tidak diperlukan.

```
int search(int depth) {
    if(depth == 0)
        return eval();
    best = -INF;
    GenMoves();
    while(MovesLeft()) {
        MakeMove();
        value = -search(depth – 1);
        UndoMove();
        if(value > best)
            best = value;
    }
    return best;
}
```

Gambar : pseudo-code algoritma *NegaMax*

Algoritma *MinMax* dan *NegaMax* diaplikasikan dalam *computer chess* dalam hal pengmabilan langkah yang akan dieksekusi. Sebagai gambaran, ketika suatu *opening* permainan catur, komputer akan men-*generate* sebuah pohon DFS lengkap yang berisi langkah-langkah legal. Selain menyimpan langkah yang mungkin, dalam setiap *node* akan disimpan suatu nilai yang telah dihitung secara *heuristic* dengan formula tertentu.

Setelah terbentuk sebuah pohon lengkap, hasil akumulasi nilai yang didapat setelah menjalankan beberapa langkah akan dibuat sebagai daun di pohon MinMax. Algoritma kemudian akan menentukan langkah mana yang akan diambil dengan algoritma *MinMax* atau *NegaMax*. Setelah didapat hasil akhir langkah bernilai maksimum yang akan dipilih, akan dirunut kembali langkah apakah yang akan menghasilkan nilai tersebut.

Kompleksitas algoritma *MinMax* tidak berbeda jauh dengan kompleksitas algoritma DFS. Namun karena penggunaannya dalam bidang *Computer chess*, waktu kalkulasinya pun menjadi sangat besar.

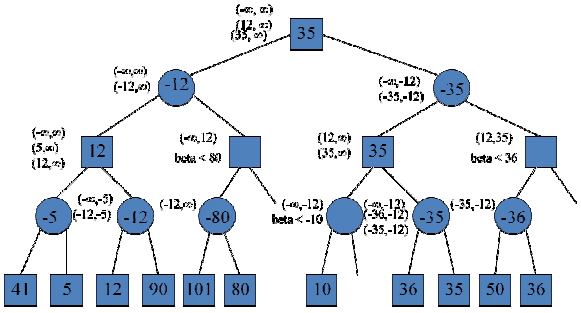
Dengan pendekatan kompleksitas secara banyaknya *node* yang dibangkitkan dan penghitungan waktu kalkulasi. Dengan struktur pembangkitan pohon yang DFS, akan terdapat b^d *node* (b : jumlah *ary* pohon, d : kedalaman pohon). Sebagai contoh adalah untuk perhitungan 10 langkah ke depan. Dengan mengasumsikan sebuah kemungkinan langkah legal dalam catur adalah 36 langkah. Akan terdapat 36^{10} *node* yang akan dibangkitkan dan harus dihitung nilainya. Dengan mengambil contoh sebuah kemampuan memproses *node* suatu superkomputer 100 ribu *nodes* per detik, akan dibutuhkan waktu kalkulasi sebesar 10 juta jam.

3.2 Algoritma *Alpha-Beta*

Algoritma *Alpha-Beta* juga merepresentasikan semua langkah kemungkinan dalam catur sebagai *node*. Metode pendekatan pembangkitannya pun hamper sama dengan algoritma *MinMax*, namun ada satu hal yang membuatnya mampu memangkas biaya kalkulasi yang cukup besar.

Bila algoritma *MinMax* lebih menggunkaan algoritma BFS sebagai metode pembangkitan *node*, algoritma *Alpha-Beta* lebih condong kea rah algoritma *Branch Bound*. Ide utama dari algoritma ini adalah jika suatu langkah sudah terbukti cukup buruk, maka tidak perlu membuang waktu untuk sesuatu yang tidak mengubah hasil akhir.

Konsep dasar yang diberikan algoritma ini adalah suatu *sentinel* atau pembatas/penjaga. Yang disebut sebagai suatu variabel bebas yang dilabeli dengan α (*Alpha*) *dan* β(*Beta*). Definisikan suatu aturan bahwa α adalah suatu batas bawah dan β adalah batas atas. inisiasikan nilai α dengan –(∞) negatif tak terhingga dan β dengan (∞)tak terhingga.



Gambar : Ilustrasi algoritma *Alpha-Beta*

Dari hasil inisiasi tersebut, untuk setiap kemunkinan masukan yang mungkin akan menyebabkan beberapa *state* yang muncul:

a. x < α : sebuah langkah buruk, karena kita sudah mempunyai suatu nilai yang lebih baik tersimpan.

b. α< x <β :nilai yang lebih baik dengan nilai yang sudah kita punya. Ganti nilai α dengan x

c.β <= x :nilai yang sangat baik sekali. *State* ini akan langsung diambil karena *state* ini akan muncul jika dan hanya jika musuh belum pernah mengambil langkah ini, selain itu juga akan menyebabkan langkah sebelumnya musuh menjadi bernilai sangat buruk. *State* ini disebut sebagai *fail-high cutoff*.

Berikut ini adalah *pseudo-code* untuk algoritma *Alpha-Beta*

```
int search(int depth                ) {
    if(depth == 0)
            return eval();
    best = -INF;
    GenMoves();
    while(MovesLeft()) {
            MakeMove();
            value = -search(depth – 1          );
            UndoMove();
            if(value >= beta)
                    return beta;
            if(value > best)
                    best = value;
            if(best > alpha)
                    alpha = best;
```

```
    }

    return best;

}
```

Gambar : *Pseudo-code* algoritma *Alpha-Beta*

Untuk aplikasi algoritma ini dalam *computer chess* dapat dilihat pada *MinMax* karena pengguanannya sama. Yang membedakan adalah algoritma ini tidak memperhitungkan kembali sebuah langkah yang terbukti lebih kecil dari nilai sementara yang kita punya sekarang.

Dengan pendekatan kompleksitas secara banyaknya *node* yang dibangkitkan dan penghitungan waktu kalkulasi. Banyaknya *node* yang akan dibangkitkan sebanyak $b^{d/2}$. (*b: ary, d:* kedalaman). Dengan asumsi yang sama dengan *MinMax*, yaitu sebanyak 36 kemungkinan langkah dan perkiraan 10 langkah ke depan, akan dibangkitkan *node* sebanyak $36^{d/2} = 6^{10}$. Dan dengan sebuah superkomputer berkecepatan proses yang sama 100 ribu *nodesi* per detik akan dibutuhkan waktu kalkulasi selama 600 detik.

4. Kesimpulan

- *Computer chess* merupakan suatu permasalahan yang sangat kompleks sehingga memerlukan optimasi perhitungan yang sangat mangkus
- Algoritma *MinMax* merupakan algoritma non-*brute forcei* pertama untuk *computer chess* yang mengawali perkembangan algoritma lainnya.
- Algoritma *Alpha-Beta* jauh lebih mangkus dalam aplikasinya di bidang *computer chess*.

REFERENSI

[1] cosmology.kaist.ac.kr/pm1/talks/chess/chess12
[2] en.wikipedia.org/wiki/Alpha-beta_pruning
[3] en.wikipedia.org/wiki/Computer_chess
[4] en.wikipedia.org/wiki/IBM_Deep_Blue
[5] en.wikipedia.org/wiki/Minimax
[6] www.omiddavid.com
[7]www.wisdom.weizmann.ac.il/~naor/COURSE/shannon_chess