PENERAPAN ALGORITMA MINIMAX DENGAN OPTIMASI MTD(*f*) PADA PERMAINAN CATUR

Anwari Ilman
(13506030)

Jurusan Teknik Informatika, Sekolah Teknik Elektro dan Informatika,
Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
e-mail: anwari.ilman@comlabs.itb.ac.id

ABSTRAK

Algoritma Minimax (juga sering disebut Minmax) adalah sebuah algoritma yang mendasari pola pikir langkah penyelesaian masalah dalam beberapa jenis permainan komputer, seperti *tic-tac-toe*, *othello*, *checkers*, catur, dll.

Pada dasarnya, algoritma Minimax sangat andal untuk menyelesaikan segala masalah dalam pencarian langkah untuk permainan komputer dengan jumlah kemungkinan penyelesaian yang kecil, seperti pada permainan *tic-tac-toe*. Tetapi, jika algoritma Minimax digunakan pada permainan dengan jumlah kemungkinan penyelesaian yang besar seperti pada permainan catur, algoritma Minimax ini memerlukan waktu yang sangat lama untuk membangun pohon penyelesaian.

Oleh karena itu, beberapa metode optimasi telah dikembangkan untuk membatasi melonjaknya jumlah simpul dalam pembangunan pohon penyelesaian. Berbagai jenis metode telah ditemukan untuk mengoptimasi algoritma Minimax, seperti *alpha-beta search*, *NegaScout*, *Null Move Search*, MTD(*f*), dll. Dengan menggunakan metode optimasi inilah proses komputasi penyelesaian masalah pada permaian catur dapat diminimalisasi. Optimasi dengan metode MTD(*f*) adalah metode yang efisien jika dibandingkan degan metode-metode yang lainnya. Makalah ini akan membahas mengenai penggunaan optimasi MTD(*f*) pada algoritma Minimax dalam permainan catur.

Kata kunci: Minimax, catur, MTD(*f*), permainan, optimasi.

1. PENDAHULUAN

Permainan catur adalah sebuah permainan satu-lawan-satu yang membutuhkan logika untuk memenangkannya. Catur sudah ada sejak bertahun-tahun lamanya dan hingga saat ini masih dimainkan orang-orang dari seluruh penjuru dunia. Sekarang ini, permainan catur sudah berkembang pesat, terbukti dengan adanya lawan baru bagi manusia dalam dunia catur, yaitu komputer. Lebih jauh lagi, permainan catur menjadi sorotan publik ketika juara dunia kejuaraan catur Garry Kasparov mengalami kekalahan melawan super komputer Deep Blue dari IBM pada bulan Mei 1997. Bahkan manusia terhebat dalam permainan catur bisa kalah ketika berhadapan dengan mesin.

Kekuatan dalam permainan catur pada manusia terletak dari strategi dalam merencanakan langkah, pemikiran jauh kedepan, intuisi, dan kreatifitas. Sedangkan komputer hanya mengandalkan kemampuan komputasi yang sangat cepat.

Kekuatan komputer yang mengandalkan proses komputasi yang sangat cepat harus didukung oleh algoritma yang memiliki efisiensi dan efektifitas yang tinggi. Untuk menyelesaikan permasalahan ini, permainan catur pada komputer menggunakan optimasi dari algoritma Minimax untuk memberikan hasil terbaik dalam waktu yang singkat.

2. ALGORITMA MINIMAX

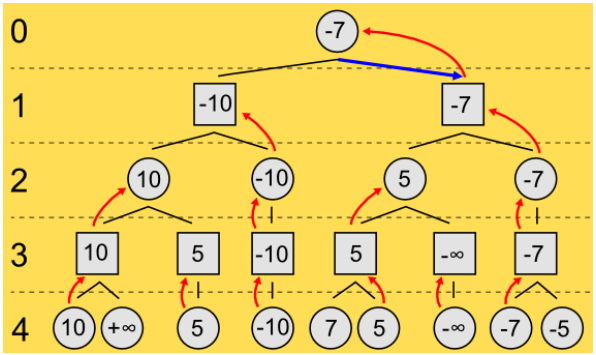
2.1 Dasar Teori

Secara teori, algoritma Minimax didefinisikan sebagai berikut:

"*Untuk setiap permainan satu-lawan satu, ada sebuah nilai yang bernilai V dan strategi yang dipilih oleh tiap pemain, sehingga: (a), Jika diberikan strategi dari pemain ke-2, maka langkah penyelesaian terbaik dari pemain pertama adalah V. Dan (b), jika diberikan strategi dari pemain pertama, maka langkah penyelesaian terbaik dari pemain kedua adalah –V*" [6]

Singkatnya, pemain pertama memberikan langkah penyelesaian yang bernilai V terhadap permainan pemain

kedua, dan sebaliknya, pemain kedua memberikan langkah peyelesaian bernilai –V. Pemikiran inilah yang mendasari asal usul penamaan algoritma Minimax, dimana pemain yang satu berjuang untuk mendapat nilai maksimum,sedangkan lawannya berjuang untuk mendapat nilai minimum.



Gambar 1. Gambar pohon yang dibangun dengan algoritma Minimax. Disini MAX diwakili aras genap, sedangkan MIN diwakili aras ganjil.

Langkah-langkah membuat algoritma Minimax adalah sebagai berikut:

1. Misalkan ada 2 pemain yang terlibat, kita namakan MAX dan MIN.
2. Lalu sebuah pohon pencarian dibangkitkan secara *depth-first-search* dari posisi awal permainan hingga akhir permainan.
3. Dari sudut pandang MAX, akan dicari posisi terakhir yang paling menguntungkan bagi MAX.

Algoritma Minimax adalah sebagai berikut:

```
MinMax (GamePosition game) {
  return MaxMove (game);
}

MaxMove (GamePosition game) {
  if (GameEnded(game)) {
    return EvalGameState(game);
  }
  else {
    best_move <- {};
    moves <- GenerateMoves(game);
    ForEach moves {
       move <-
MinMove(ApplyMove(game));
       if (Value(move) >
Value(best_move)) {
          best_move <- move;
       }
    }
    return best_move;
  }
}
```

```
MinMove (GamePosition game) {
  best_move <- {};
  moves <- GenerateMoves(game);
  ForEach moves {
     move <-
MaxMove(ApplyMove(game));
     if (Value(move) >
Value(best_move)) {
        best_move <- move;
     }
  }

  return best_move;
```

Yang terjadi pada algoritma diatas adalah, MAX akan mengambil nilai maksimum dari pohon pencarian yang dibangkitkan pada simpul terakhir. Sebaliknya, MIN akan menangkis serangan MAX dengan mengambil nilai minimum pada posisi akhir permainan, yang akan meminimalisasi serangan MAX.

2.2 Implementasi Algoritma Minimax

Untuk lebih jelasnya, kita misalkan ada sebuah permainan sederhana yang dengan hanya ada satu langkah untuk tiap pemain dengan kemungkinan situasi seperti ini:

Pergerakan MAX	Pergerakan MIN	Nilai Evaluasi
A	C	12
A	D	-2
B	C	5
B	D	6

Tabel 1. Contoh nilai akhir pergerakan MAX dan MIN pada pohon Minimax.

Fungsi evaluasi berlaku untuk posisi akhir papan, yang berupa kombinasi langkah dari MIN dan MAX.

MAX berasumsi jika MIN akan bermain dengan baik. Maka dari itu, MAX tahu jika dia melakukan pergerakan A, maka MIN akan membalasnya dengan melakukan pergerakan D, yang mengakibatkan nilai evaluasi -2 (kemenangan bagi MIN). Bagaimanapun juga, jika MAX melakukan gerakan B, dia pasti akan menang karena pergerakan MIN yang terbaik hanya akan menghasilkan nilai evaluasi terbaik sebesar 5 saja. Jadi, dengan menggunakan algoritma Minimax, MAX akan selalu memilih untuk melakukan langkah B, walaupun dia sebenarnya akan mendapatkan kemenangan yang lebih baik jika melakukan A dan MIN melakukan kesalahan dengan memilih langkah C.

2.3 Kelemahan Algoritma Minimax

Dengan membangkitkan pohon pencarian ini, komputer akan selalu mendapatkan penyelesaian terbaik untuk setiap langkahnya. Tetapi, masalah muncul ketika jumlah input yang dimasukkan menjadi banyak. Bayangkan saja, untuk posisi pertama dalam permainan catur dimana hanya terdapat kemungkinan untuk bergerak bagi 8 pion dan 2 kuda, akan terdapat sebanyak 8*2 + 2*2 = 20 kemungkinan. Hal ini berarti pada simpul awal akan didapatkan 20 sub-simpul sebagai langkah penyelesaian pada aras (*level*) ke 1. Jika komputer melakukan hal yang sama untuk menganalisis langkah selanjutnya (berarti komputer melakukan pembangkitan pohon untuk simpul pada aras ke 1) maka akan dilakukan 20 pembangkitan dimana pada tiap pembangkita akan dilakukan puluhan pembangkitan lain untuk aras ke 2. Demikian seterusnya hingga algoritma mencapai aras terakhir permainan, skakmat.

Berarti, usaha yang dilakukan akan tumbuh secara dramatis mengikuti:

- Jumlah kemungkinan posisi penyelesaian untuk setiap pemain, dinamakan *branching factor*. Sering dinotasikan sebagai B.
- Kedalaman aras pada pembangunan pohon pencarian, sering dinotasikan sebagai d. Biasa disebut juga '*ply*', yang berarti sebuah langkah yang dilakukan oleh pemain.

Jika komputer hanya mengandalkan algoritma Minimax ini, proses membangkitkan simpul akan menjadi sangat lama. Kompleksitas dari algoritma Minimax ini adalah eksponensial sebesar $O(B^d)$ [5]. Dengan nilai B untuk catur rata-rata adalah 35, dan jika kita akan menentukan 9 langkah kedepan, diperlukan sekitar 50 juta kemungkinan untuk dieksplorasi. Hal ini tentunya membutuhkan usaha yang sangat keras untuk mengecek seluruh kemungkinan. Tentu saja diperlukan waktu yang lama untuk menganalisis seluruh kemungkinan ini dalam sebuah permainan catur jika hanya mengandalkan algoritma Minimax.

3. OPTIMISASI ALGORITMA MINIMAX

Untuk menyiasati banyaknya simpul yang dibangkitkan dalam permainan catur dengan menggunakan algoritma Minimax, perlu dibuat sebuah strategi untuk membatasi ruang lingkup komputasi dari algoritma Minimax. Kita tidak dapat begitu saja menghilangkan algoritma Minimax dari permainan catur dikarenakan ketidakmangkusan dan ketidaksangkilan dari Minimax, tapi tentu saja ada cara untuk membuat algoritma ini lebih andal.

3.1 Algoritma MTD(*f*)

Algoritma MTD(*f*) adalah sebuah algoritma optimasi Minimax baru yang lebih sederhana dan lebih sangkil daripada beberapa pendahulunya[4]. Nama dari algoritma ini adalah kependekan dari MTD(n,f), yang disingkat dari *Memory-enhanced Test Driver with node n and value f*. MTD adalah nama dari sekumpulan driver program yang mencari pohon Minimax menggunakan pemanggilan *zero-window AlphaBetaWithMemory*.

Dalam beberapa percobaan permainan komputer seperti catur, *othello*, dan *checkers*, algoritma ini mempunyai performa rata-rata lebih baik daripada *Negascout* (variasi dari *Alphabeta* yang diimplementasikan dalam hampir semua permainan catur, *checkers*, dan *othello*). Salah satu program catur terkuat, *Cilkchess* (http://theory.lcs.mit.edu/~cilk/chess.html) milik MIT yang menggunakan metode komputasi pararel, juga menggunakan MTD(*f*) sebagai algoritma pencariannya menggantikan *Negascout* yang digunakan oleh program catur pendahulunya, *StarSocrates*.

3.2 Implementasi Algoritma MTD(*f*)

MTD(*f*) 'hanya' terdiri dari 10 baris kode saja, yaitu seperti berikut:

```
function MTDF(root, f, d)
    g := f
    upperBound := +•
    lowerBound := -•
    while lowerBound < upperBound
        if g = lowerBound then
            • := g+1
        else
            • := g
        g :=
AlphaBetaWithMemory(root, •-1, •, d)
        if g < • then
            upperBound := g
        else
            lowerBound := g
    return g
```

Algoritma MTD(*f*) diatas memanggil fungsi *AlphaBetaWithMemory* berkali-kali dengan metode *zero-window* pencarian *Alpha-beta*, tidak seperti *Negascout* yang menggunakan pencarian *wide-window*. Pemanggilan *AlphaBetaWithMemory* mengembalikan batas dari nilai evaluasi Minimax. Batas dari nilai itu kemudian disimpan dalam *upperbound* (batas atas) dan *lowerbound* (batas bawah), membentuk sebuah interval yang melingkupi nilai Minimax yang sebenarnya pada pencarian dengan

kedalaman tertentu. Positif dan negatif tak hingga adalah kependekan dari nilai diluar interval pada daun pohon. Ketika batas atas dan batas bawahnya bernilai sama atau batas bawah telah melampaui nilai batas atas, maka nilai Minimax telah ditemukan.

Efisiensi dari MTD(*f*) berasal dari pencarian *Alpha-beta* dengan *zero-window*, dan menggunakan sebuah nilai batas yang baik (variabel *beta*) untuk melakukan pencarian *zero-window* tersebut. Dalam versi yang sebelumnya, *Alpha-beta* dipanggil dengan pencarian wide-window seperti ini *AlphaBeta*(*root*, -INFINITY, +INFINITY, *depth*), yang memastikan nilai kembaliannya berada dalam interval *alpha* dan *beta*. Sedangkan dalam algoritma MTD(*f*), digunakan pencarian *zero-window*, jadi untuk setiap pemanggilan *Alpha-beta* akan mengembalikan batas bawah dan batas atas dari nilai Minimax berturut-turut. Pencarian dengan *zero-window* memberikan lebih banyak jalan pintas, tetapi lebih sedikit informasi –hanya batas dari nilai minimax saja. Untuk menyiasati hal itu MTD(*f*) perlu memanggil *Alpha-beta* beberapa kali, untuk mendekati nilai itu. Imbas dari pemanggilan *Alpha-beta* secara berulang-ulang dapat dihilangkan dengan menggunakan versi dari *Alpha-beta* yang menyimpan nilai simpul dalam memori.

Agar algoritma MTD(*f*) dapat berjalan dengan baik, maka diperlukan "tebakan pertama" sebagai pengarah untuk menemukan nilai Minimax yang baik. Semakin baik tebakan pertama, maka algoritma tersebut akan semakin efisien, dan diperlukan lebih sedikit pengulangan. Jika kita memberikan tebakan nilai Minimax pada MTD(*f*), dia hanya akan melakukan dua pengulangan, yang satu untuk menentukan batas atas dari nilai *x*, dan yang satunya lagi untuk menemukan batas bawah dari nilai tersebut.

3.3 Algoritma AlphaBetaWithMemory

Sebagai catatan, MTD(*f*) memanggil fungsi Alpha-Beta yang menyimpan nilai simpul-simpulnya dalam memori, dan mengembalikan nilainya untuk pencarian kembali. Untuk membuat MTD(*f*) sangkil, maka fungsi Alpha-beta harus menyimpan nilai simpul yang telah dicari.

Berikut ini adalah kode dari *AlphaBetaWithMemory*:

```
function AlphaBetaWithMemory(n :
node_type; alpha , beta , d : integer)
: integer;
      if retrieve(n) == OK then /*
      Melihat tabel transposisi */
            if n.lowerbound >= beta
            then return n.lowerbound;
            if n.upperbound <= alpha
            then return n.upperbound;
            alpha := max(alpha,
            n.lowerbound);
            beta := min(beta,
            n.upperbound);
      if d == 0 then g :=
      evaluate(n); /* simpul daun */
      else if n == MAXNODE then
            g := -INFINITY; a :=
            alpha; /* menimpan nilai
            alpha yang sebenarnya */
            c := firstchild(n);
            while (g < beta) and (c
            != NOCHILD) do
                  g := max(g,
                  AlphaBetaWithMemory
                  (c, a, beta, d -
                  1));
                  a := max(a, g);
                  c :=
                  nextbrother(c);
      else /* n adalah sebuah
      MINNODE */
            g := +INFINITY; b :=
            beta; /* save original
            beta value */
            c := firstchild(n);
            while (g > alpha) and (c
            != NOCHILD) do
                  g := min(g,
                  AlphaBetaWithMemory
                  (c, alpha, b, d -
                  1));
                  b := min(b, g);
                  c :=
                  nextbrother(c);
      /* Tabel transposisi menyimpan
      nilai simpul */

      if g <= alpha then n.upperbound
      := g; store n.upperbound;

      if g >  alpha and g < beta then
            n.lowerbound := g;
            n.upperbound := g; store
            n.lowerbound,
            n.upperbound;

if g >= beta then n.lowerbound := g;
store n.lowerbound;
return g;
```

Tabel transposisi yang digunakan berguna untuk menyimpan dan mengambil panggilan. `Retrieve` berfungsi untuk memastikan jika sebuah nilai terdapat

dalam tabel, maka nilai itu akan langsung digunakan, bukan dilanjutkan dengan mencarinya. Store berfungsi untuk menyimpan nilai yang ada. Dalam algoritma ini, kita juga akan menyimpan nilai langkah terbaik kedalam tabel transposisi.

4. KESIMPULAN

Untuk setiap permainan satu-lawan-satu, hampir selalu digunakan algoritma Minimax dalam permainan komputer. Proses pembangunan pohon pencarian Minimax dilakukan dengan metode depth-first-search. Algoritma Minimax mampu menganalisis segala kemungkinan posisi permainan untuk menghasilkan keputusan yang terbaik. Tetapi, dengan kelebihan yang seperti ini, algoritma Minimax tidak sangkil untuk memroses data dengan ukuran masukan yang besar, karena proses untuk membangun pohon pencarian dangan algoritma ini memiliki kompleksitas algoritma eksponensial. Maka dari itu, diperlukan optimasi dalam algoritma ini agar tidak semua simpul dibangkitkan.

Jenis optimasi dalam permainan catur dengan algoritma Minimax ada bermacam-macam, tetapi dalam makalah ini hanya dibahas mengenai optimasi MTD(*f*) karena algoritma MTD(*f*) diklaim sebagai algoritma terbaik dibandingkan dengan algoritma-algoritma pendahulunya. Dengan menggunakan algoritma MTD(*f*) ini, proses pencarian nilai Minimax berlangsung lebih cepat.

REFERENSI

[1] Abdul Gapur, "Aplikasi Struktur Data The Minimax Game Tree pada Permainan Catur", Teknik Informatika ITB, 2007.

[2] Khoirush Sholih Ridhwaana Akbar, "Algoritma Minimax dalam Pengambilan Keputusan pada Permainan Tic-tac-toe", Teknik Informatika ITB, 2007.

[3] Rainer Feldmann, "Computer Chess: Algorithm and Heuristics for a Deep Look into The Future *", University of Paderborn, Jerman. 1-3.

[4] Aske Plaat: MTD(f), a new chess algorithm
www.cs.vu.nl/~aske/mtdf.html
diakses pada tanggal 14 Mei 2008

[5] Chess Programming Part IV: Basic Search
http://www.gamedev.net/reference/articles/article1171.asp
diakses pada tanggal 14 Mei 2008

[6] Minimax – Wikipedia, The Free Encyclopedia
en.wikipedia.org/wiki/Minimax
diakses pada tanggal 15 Mei 2008

[7] Minimax explained – AI Depot
ai-depot.com/articles/minimax-explained/
diakses pada tanggal 14 Mei 2008

[8] MTD-f – Wikipedia, The Free Encyclopedia
en.wikipedia.org/wiki/MTD-f
diakses pada tanggal 15 Mei 2008