# KOMBINASI *GREEDY*, *MINIMAX*, DAN *ALPHA-BETA PRUNING* UNTUK PERMAINAN REVERSI

**I.Y.B. Aditya Eka Prabawa W.**

Laboratorium Ilmu dan Rekayasa Komputasi, Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganeca 10, Bandung 40132, Indonesia
e-mail: aditya_eka@students.itb.ac.id

## ABSTRAK

**Makalah ini adalah sebuah *technical report* dari pengembangan algoritma *Greedy* dengan menggunakan algoritma *Branch and Bound* untuk permainan *Reversi* (*Othello*). Pengembangan yang dilakukan berdasarkan pada salah satu tugas pemrograman mata kuliah IF3051 – Strategi Algoritma, dimana pada tugas tersebut diaplikasikan algoritma *Greedy* pada permainan *Reversi*. Pengembangan dengan *Branch and Bound* menggunakan algoritma *MiniMax* yang dioptimasi dengan *Alpha-Beta Pruning*. Dalam makalah ini akan dibahas secara singkat strategi *Greedy by Number* yang akan digunakan sebagai pembanding. Pembahasan akan dilanjutkan dengan prinsip algoritma *Branch and Bound* secara umum, prinsip dan ilustrasi algoritma *MiniMax* dan optimasinya dengan *Alpha-Beta Pruning*, serta penggunaannya dalam permainan Reversi. Pembahasan kedua algoritma tersebut akan dibantu dengan penyajian *pseudo-code* dan Pohon Permainan (*Game Tree*) yang bersesuaian. Pada bagian akhir makalah, akan disajikan hasil pengujian kinerja algoritma *Greedy* dengan strategi *Greedy by Number* melawan pengembangannya, yaitu algoritma *Branch and Bound* (*Greedy by Number* + *MiniMax* + *Alpha-Beta Pruning*) pada program permainan *Reversi*.**

**Kata kunci:** *Reversi*, *Greedy*, *Branch and Bound*, *MiniMax*, *Alpha-Beta Pruning*.

## 1. PENDAHULUAN

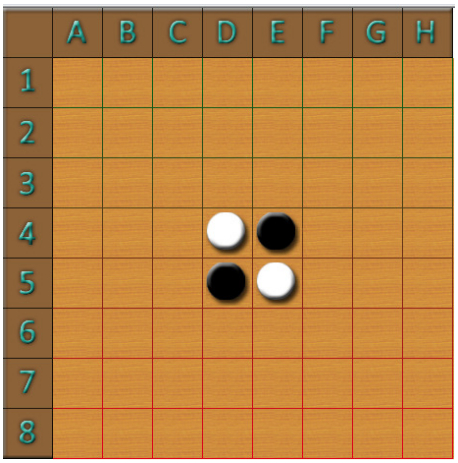
Permainan Reversi sebenarnya lebih dikenal luas dengan nama Othello. Namun karena Othello adalah sebuah merek dagang, maka pengarang memilih nama Reversi. Permainan Reversi ditemukan pada tahun 1883 oleh dua orang berkebangsaan Inggris, yaitu Lewis Waterman dan John Mollet. Permainan ini menjadi populer baik karena kesederhanaannya maupun karena kompleksitasnya. Namun demikian, walaupun dapat dibilang kompleks, saat ini Reversi telah menjadi salah satu permainan papan yang dapat diselesaikan oleh komputer.

Permainan Reversi dimainkan pada papan berpetak 8 x 8. Pada awal permainan empat petak tengah diisi oleh dua keping hitam dan dua keping putih. Tidak seperti pada permainan catur, pada permainan Reversi pemain keping hitam selalu bergerak terlebih dahulu. Tujuan permainan adalah untuk meletakkan keping-keping pada papan sedemikian rupa sehingga satu atau lebih keping lawan 'terjepit' di antara dua keping pemain. Jika keping lawan terjepit mereka dibalikkan ke warna pemain. Permainan berlanjut selama setidaknya salah satu pemain masih dapat bergerak, dan pemain tidak memiliki langkah jika tidak ada petak yang dapat diisi dengan keping warnanya. Permainan berakhir jika tidak ada lagi langkah yang dapat dilakukan, pada saat permainan berakhir pemain dengan nilai tertinggi menang (jumlah keping terbanyak).

Ada banyak sekali teknik komputasi yang dapat digunakan untuk menyelesaikan permainan Reversi. Diantaranya adalah: Pemilihan Langkah secara Acak (*Random*), Algoritma *Greedy*, Algoritma *Branch and Bound* (Algoritma *Alpha-Beta Pruning*), Metode Heuristik (*Open Book*) dimana langkah-langkah yang mungkin disimpan pada sebuah Basis Pengetahuan (*Knowledge Base*), dan gabungan semua teknik-teknik tersebut.

Pada salah satu tugas pemrograman mata kuliah IF3051 – Strategi Algoritma, telah dibuat sebuah program permaianan Reversi yang mengaplikasikan algoritma *Greedy*. Algoritma *Greedy* ini akan dikembangkan dengan menggunakan salah satu algoritma *Branch and Bound*, yaitu algoritma *MiniMax* yang dioptimasi dengan *Alpha-Beta Pruning*. Disebut pengembangan karena walaupun diaplikasikan algoritma yang lebih pintar, masih terdapat identitas *Greedy* yang tetap digunakan dalam algoritma *Branch and Bound* hasil pengembangan. Dengan kata lain, algoritma pengembangan akan mengkombinasikan tiga teknik penyelesaian Reversi, yaitu *Greedy by Number*, *MiniMax*, dan *Alpha-Beta Pruning*.

Dengan pengembangan ini diharapkan program Reversi dapat menjadi lebih pintar dan lebih sulit dikalahkan daripada program Reversi yang hanya mengaplikasikan algoritma *Greedy*. Selain itu, dengan mengaplikasikan algoritma *Branch and Bound* ini, pengaturan tingkat kesulitan pada program dapat diatur dengan lebih dinamis.




**Gambar 1. Kondisi papan Reversi pada awal permainan**

## 2. REVERSI DENGAN ALGORITMA *GREEDY*

Strategi *Greedy* yang digunakan adalah *Greedy by Number*. Pada setiap langkah permainan *Greedy Engine* memilih langkah penempatan keping yang akan menjepit keping lawan dengan jumlah terbanyak. Langkah-langkah *Greedy* yang digunakan adalah sebagai berikut, pada setiap langkah permainan, lakukan:

1. Pindai seluruh papan, pada setiap petak periksa apakah petak tersebut boleh diisi oleh keping dengan warna tertentu (Terdapat sebuah fungsi khusus untuk melakukan pemeriksaan ini).
2. Setiap petak yang boleh diisi dimasukkan ke dalam sebuah senarai (*list*) yang menyimpan posisi petak dalam papan dan nilai petak tersebut (Yang dimaksud nilai petak disini adalah jumlah keping lawan yang dapat dijepit dengan peletakkan keping pemain pada petak tersebut).
3. Urutkan senarai yang didapatkan berdasarkan nilai petak yang terurut mengecil (Petak-petak dengan nilai terbesar akan terletak pada awal senarai).
4. Ambil petak yang pertama pada senarai sebagai langkah selanjutnya (Bila terdapat lebih dari satu petak dengan nilai yang sama, petak akan dipilih secara acak / *random* untuk memastikan alur permainan yang lebih dinamis).

Dengan menggunakan prinsip *Greedy*, program Reversi masih sangat mudah dikalahkan, karena *Greedy Engine* hanya mengambil sebuah solusi yang dianggap optimal pada suatu langkah tanpa mempertimbangkan konsekuensi pada langkah-langkah selanjutnya. Selain itu, *Greedy by Nnumber* adalah salah satu strategi *Greedy* yang paling sederhana dan kurang pintar. Masih banyak strategi *Greedy* yang lebih pintar, namun itu di luar lingkup pembahasan makalah ini. Pembahasan akan dilanjutkan dengan teknik penyelesaian Reversi yang lebih pintar, yaitu dengan menggunakan kombinasi algoritma *Greedy* dan algoritma *Branch and Bound*.

## 2. REVERSI DENGAN ALGORITMA *BRANCH AND BOUND*

### 2.1 Prinsip Algoritma *Branch and Bound*

Algoritma *Branch and Bound* merupakan metode pencarian di dalam ruang solusi secara sistematis. Ruang solusi diorganisasikan ke dalam pohon ruang status. Algoritma *Branch and Bound* membangun ruang solusi dengan skema BFS (*Breadth-First Search*). Untuk mempercepat pencarian ke simpul solusi, maka setiap simpul diberi nilai ongkos (*cost*).

Pada algoritma *Branch and Bound*, pencarian ke simpul solusi dapat dipercepat dengan memilih simpul hidup berdasarkan nilai ongkos. Setiap simpul hidup diasosiasikan dengan sebuah ongkos yang menyatakan nilai batas (*bound*).

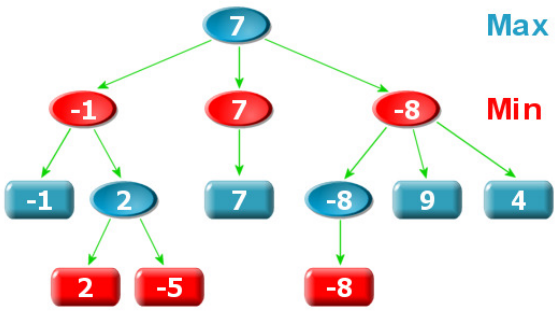
### 2.2 Algoritma *MiniMax*

MiniMax (*Minimize the Maximum Possible Loss*) adalah algoritma yang "melihat beberapa langkah ke depan", dengan cara melakukan pencarian pada Pohon Permainan (*Game Tree*), yang menyimpan posisi-posisi papan. Pada Pohon Permainan, setiap simpul melambangkan posisi papan. Anak dari setiap simpul *S* adalah posisi-posisi berbeda yang dapat dihasilkan dari satu langkah pada *S*, yang dilakukan oleh pemain yang mendapat giliran saat *S*. Akar dari pohon ini adalah posisi awal permainan, atau posisi saat program akan mencari langkah selanjutnya. Algoritma *MiniMax* diimplementasikan secara rekursif.

Pada permainan Reversi, algoritma *MiniMax* digunakan untuk menemukan nilai akurat untuk sebuah posisi papan. Algoritma *MiniMax* mengasumsikan bahwa kedua pemain selalu mengambil langkah terbaik pada setiap posisi. Dengan asumsi ini, dua pengamatan menghasilkan algoritma *MiniMax*. Didapatkan analisis akurat untuk daun-daun, dan nilai dari sebuah simpul dapat didapatkan dengan akurat dari nilai anak-anaknya. Nilai daun dapat ditemukan dengan mudah, karena pada posisi inilah basis rekursi, dan nilainya didapatkan dengan menggunakan fungsi analisis / evaluasi.

Setiap simpul yang bukan daun merupakan keputusan yang diambil oleh pemain yang mendapat giliran pada aras simpul tersebut. Pemain yang melangkah pada simpul tersebut harus menentukan simpul anak yang mana yang harus dipilih sebagai posisi papan selanjutnya. Bila pemain biru, ia harus memilih anak dengan nilai terbesar. Dan pemain merah harus memilih posisi yang meminimasi nilai simpulnya. Dengan cara ini bisa didapatkan nilai untuk setiap simpul, dan dengan

demikian dapat ditemukan langkah terbaik yang dapat dilakukan dari setiap simpul. Gambar berikut akan mengilustrasikan algoritma *MiniMax*.




**Gambar 2. Prinsip Algoritma MiniMax**

Membangun seluruh ruang solusi dengan Pohon Permainan sampai posisi akhir papan permainan tidak mungkin dilakukan dalam Reversi. Oleh karena itu digunakan fungsi analisis / evaluasi. Karena tidak dapat dibangun seluruh Pohon Permainan, maka kedalamam pohon pencarian harus dibatasi dan simpul-simpul di bawah batas kedalaman dianggap daun, dan nilainya dicari dengan menggunakan fungsi analisis / evaluasi. Walapun demikian, membangun pohon yang semakin dalam akan memungkinkan program untuk melihat lebih banyak langkah ke depan, dan dapat mengambil langkah yang lebih baik, dengan harga waktu pemrosesan yang meningkat.

*Pseudo-code* algoritma *MiniMax* diberikan sebagai berikut:

```
function MiniMax(input M : Matrix, input
depth.Player : integer) → integer
{
GameOver() adalah fungsi yang memeriksa apakah
kondisi papan permainan telah menggambarkan akhir
permainan.
Evaluation() adalah fungsi yang menghitung nilai
sebuah simpul daun.
CanPlay() adalah fungsi yang memeriksa apakah
seorang pemain dapat bergerak.
ValidMoves() adalah fungsi yang menghasilkan
senarai langkah yang dapat dilakukan seorang
pemain.
Sort() adalah fungsi yang mengurutkan isi
senarai.
CopyMatrix() adalah fungsi yang menyalin isi
Matriks.
Move() adalah fungsi yang mengeksekusi langkah
seorang pemain pada papan permainan.
EmptyMove() adalah fungsi yang memeriksa apakah
sebuah langkah null atau tidak.
}
{ KAMUS }
MaxDepth : integer
otherPlayer,currentPlayer : integer
Value,bestValue : integer
i : integer
moves : ListMove
N : Matrix
bestMove : Move

{ ALGORITMA }
if(GameOver(M) or (depth > MaxDepth)) then
      → Evaluation(B,currentPlayer)
if(not CanPlay(Player)) then
      → MiniMax(M,depth,otherPlayer)
moves ← ValidMoves(Player)
if(Player = currentPlayer) then
      bestValue ← -999999999
else
      bestValue ← 999999999
i traversal [1..moves.Count]
      N ← CopyMatrix(M)
      Move(N,moves[i].x,moves[i].y,Player)
      Value ← MiniMax(N,depth+1,otherPlayer)
      if(Player = currentPlayer) then
        if(Value > bestValue) then
              bestValue ← Value
              bestMove ← moves[i]
      else
        if(Value < bestValue) then
              bestValue ← Value
              bestMove ← moves[i]
if(not EmptyMove(bestMove)) then
      Move(M,bestMove.x,bestMove.y,Player)
→ bestValue
```

`MaxDepth` adalah sebuah konstanta yang menandai batas kedalaman yang akan dievaluasi. Fungsi `MiniMax` ini dipanggil dengan nilai `depth` awal = 0. Nilai yang dikembalikan adalah nilai simpul yang melambangkan keadaan papan permainan. Untuk mencari langkah terbaik untuk setiap posisi papan, pilih saja anak simpul yang memiliki nilai terbaik (minimum atau maksimum, tergantung warna pemain).

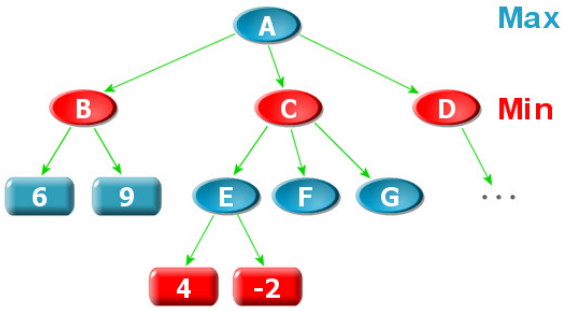
Pada basis rekursi, nilai sebuah simpul daun dihitung dengan menggunakan fungsi evaluasi. Fungsi evaluasi memberikan nilai pada sebuah posisi papan. Bila nilai hasil fungsi evaluasi tinggi, artinya posisi papan tersebut menguntungkan pemain *Maximizer* (dalam contoh ini pemain warna biru), bila nilai hasil fungsi evaluasi rendah, artinya posisi papan itu akan menguntungkan pemain lawan (*Minimizer* yang dalam contoh ini pemain warna merah). Dengan demikian, bila posisi papan tersebut ternyata memenangkan pemain biru, fungsi evaluasi akan menghasilkan nilai yang dianggap tak hingga, dan bila posisi papan tersebut ternyata memenangkan pemain merah, fungsi evaluasi akan menghasilkan nilai yang dianggap minus tak hingga. Kondisi papan permainan yang imbang memiliki nilai 0.

Dapat dilihat bahwa algoritma *MiniMax* sendiri belum dapat dikatakan sebagai algoritma *Branch and Bound*. Algoritma *MiniMax* berperilaku layaknya metode BFS murni, dimana semua anak simpul pada aras tertentu dibangkitkan tanpa batas secara melebar tanpa memperhitungkan ongkos (*cost*) dan batas (*bound*).

## 2.3 Algoritma *Alpha-Beta Pruning*

Dengan algoritma *MiniMax* saja, tidak dapat dilakukan pencarian yang dalam dengan cepat. Pada setiap langkah permainan Reversi bisa terdapat lebih dari 10 langkah yang mungkin dari setiap posisi papan, dengan demikian jumlah simpul yang harus dievaluasi meningkat secara eksponensial setiap peningkatan kedalaman aras. Oleh karena itu, perlu dilakukan optimasi dari teknik pencarian yang dilakukan.

*Alpha-Beta Pruning* adalah algoritma pencarian yang mengurangi secara drastis jumlah simpul yang dibangkitkan untuk dievaluasi pada pohon pencarian dengan algoritma *MiniMax*. (*Pruning* adalah eliminasi simpul yang tidak perlu diproses saat pencarian).




**Gambar 3. Ilustrasi *Alpha-Beta Pruning***

Dari Gambar 3 di atas, pada saat program mencari nilai A, B dievaluasi dan didapatkan bahwa nilainya adalah 6 (minimum dari 6 dan 9). Kemudian C dievaluasi. Dari sana didapatkan bahwa nilai E adalah 4 (maksimum dari 4 dan -2). Karena C adalah *Minimizer*, dapat diketahui bahwa nilai C ≤ 4. Oleh karena itu, C tidak akan dipilih oleh A sebagai maksimum, karena A telah memiliki anak B dengan nilai 6, yang tentunya akan lebih besar daripada semua nilai C yang mungkin. Dengan demikian, mengevaluasi F dan G tidak perlu dilakukan, dan proses evaluasi C dapat dihentikan (*pruned*). Dan evaluasi dilanjutkan ke simpul D.

Hal yang sama dapat dilakukan bila misalnya A bukan *Maximizer* melainkan *Minimizer*. Secara umum dapat didefinisikan batas bawah (*lower bound* / *alpha*) dan batas atas (*upper bound* / *beta*) dari nilai sebuah simpul. Salah satu batas ini (tergantung warna pemain) tidak berubah dalam simpul, dan bila nilai simpul tersebut melewati batas, simpul tersebut dapat dieliminasi. Batas yang satunya lagi berubah bila nilai simpul tersebut diubah.

Berikut ini adalah *pseudo-code* algoritma *Alpha-Beta Pruning* yang dikembangkan dari algoritma *MiniMax* yang diberikan sebelumnya:

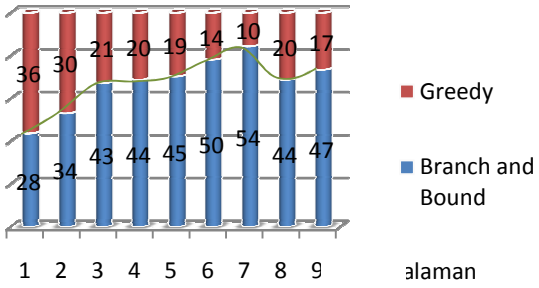
```
function AlphaBeta(input M : Matrix, input
depth.Player,alpha,beta : integer) → integer
{
Fungsi-fungsi yang digunakan disini sama dengan
pada algoritma MiniMax.
}
{ KAMUS }
MaxDepth : integer
otherPlayer,currentPlayer : integer
Value : integer
i : integer
moves : ListMove
N : Matrix
bestMove : Move

{ ALGORITMA }
if(GameOver(M) or (depth > MaxDepth)) then
      → Evaluation(B,currentPlayer)
if(not CanPlay(Player)) then
      →
AlphaBeta(M,depth,otherPlayer.alpha.beta)
moves ← ValidMoves(Player)   ← Greedy
Sort(moves)
i traversal [1..moves.Count]
      N ← CopyMatrix(M)
      Move(N,moves[i].x,moves[i].y,Player)
      Value ←
      AlphaBeta(N,depth+1,otherPlayer,alpha,beta)
      if(Player = currentPlayer) then
        if(Value > alpha) then
                alpha ← Value
                bestMove ← moves[i]
        if(alpha >= beta) then
                → alpha
      else
        if(Value < beta) then
                beta ← Value
                bestMove ← moves[i]
        if(alpha >= beta) then
                → beta
if(not EmptyMove(bestMove)) then
      Move(M,bestMove.x,bestMove.y,Player)
if(Player = currentPlayer) then
      → alpha
else
      → beta
```

Penggunaan `alpha >= beta` dan tidak `alpha > beta`, akan sedikit meningkatkan kecepatan komputasi. Fungsi `AlphaBeta` ini dipanggil dengan nilai `depth` awal = 0, `alpha` = -999999999, dan `beta` = 999999999. Secara teoretis, bila diimplementasikan dengan benar, *Alpha-Beta Pruning* akan memungkinkan program untuk mengevaluasi dua sampai tiga aras lebih banyak pada periode waktu yang sama dengan pencarian menggunakan *MiniMax* murni.

## 3. PENGUJIAN: *GREEDY VS. BRANCH AND BOUND*

Pada pengujian yang dilakukan dalam program Reversi, algoritma *Branch and Bound* yang merupakan kombinasi antara *Greedy by Number*, *MiniMax*, dan *Alpha-Beta*

*Pruning* dihadapkan dengan algorima *Greedy* murni dengan strategi *Greedy by Number*. Hasil pengujian dengan sampel 54 kali permainan diberikan dalam diagram berikut ini:




**Gambar 4. Perbandingan Nilai *Greedy* dan *Branch and Bound* dengan berbagai kedalaman**

Kedalaman Pohon Permainan pada algoritma *Branch and Bound* yang diuji merepresentasikan jumlah "langkah ke depan" yang dievaluasi oleh algoritma tersebut. Dari sampel 54 kali permainan, didapatkan bahwa secara rata-rata, jumlah keping yang didapatkan oleh algoritma *Greedy* murni (*Greedy by Number*) lebih unggul daripada algoritma *Branch and Bound* (kombinasi *Greedy by Number*, *MiniMax*, dan *Alpha-Beta Pruning*) dengan kedalaman 1 (melihat 1 langkah ke depan). Namun demikian rata-rata jumlah keping yang didapatkan oleh algoritma *Branch and Bound* cenderung meningkat seiring dengan meningkatnya batas kedalaman pohon pencarian (melihat lebih jauh ke depan).

Dari hasil pengujian juga ditemukan bahwa penyelesaian Reversi dengan pencarian menggunakan *Branch and Bound* berkedalaman minimal 3 selalu mengalahkan *Greedy*. Juga perlu dicatat bahwa *response time* program yang menjalankan *Branch and Bound* kombinasi dari algoritma *Greedy by Number*, *MiniMax*, dan *Alpha-Beta Pruning* masih terasa wajar sampai dengan kedalaman 7 aras. Pada kedalaman 8 aras, *response time* program mulai berkurang, dan pada kedalaman 9 aras, komputasi 1 langkah bisa mencapai 30 detik. Sedangkan komputasi algoritma *Greedy by Number* murni tentunya selalu konstan dan terasa cepat. (Komputasi menggunakan prosesor Intel Core Duo T2500 2x2.0 GHz).

## 4. KESIMPULAN

Pada permainan Reversi, algoritma *MiniMax* digunakan untuk secara sistematis mencari langkah-langkah ke depan untuk menentukan langkah terbaik saat ini. Dimulai dengan keadaan posisi papan saat ini sebagai simpul akar, simpul anak dibangkitkan untuk setiap langkah berikutnya yang boleh dilakukan. Pembangkitan simpul anak ini berulang terus secara rekursif sampai pohon pencarian mencapai kedalaman maksimal yang ditentukan atau konsidi posisi papan sudah mencapai akhir permainan. Setiap kali sebuah simpul dibangkitkan, setiap anaknya dievaluasi. Semakin tinggi nilai sebuah simpul, artinya langkah yang direpresentasikan oleh simpul tersebut semakin bernilai bagi pemain. Semakin rendah nilai sebuah simpul, artinya langkah yang direpresentasikan oleh simpul tersebut semakin bernilai bagi pemain lawan. Nilai-nilai ini akan diambil oleh simpul orang tuanya. Pada giliran pemain, simpul orang tua akan mengambil simpul anak yang maksimal. Pada giliran pemain lawan, simpul orang tua akan mengambil simpul anak yang minimal. Tindakan ini dilakukan dengan asumsi, pemain akan selalu mengambil langkah yang memaksimalkan nilainya, dan pemain lawan akan selalu mengambil langkah yang meminimalkan nilai pemain. Ketika simpul anak memilih simpul yang maksimal, langkah itulah yang diambil untuk dilaksanakan.

Algoritma *MiniMax* dapat dioptimasi dengan *Alpha-Beta Pruning*. *Alpha-Beta Pruning* mengurangi waktu komputasi secara cukup signifikan tanpa mengubah hasil pencarian. *Alpha-Beta Pruning* tidak membangkitkan dan mengevaluasi simpul-simpul yang pada akhirnya memang tidak akan dipilih. Untuk membuat *pruning* yang lebih efisien, langkah-langkah yang akan menjepit keping lawan terbanyak dibangkitkan dan dievaluasi terlebih dahulu (diadopsi dari prinsip *Greedy by Number*). Dengan demikian, probabilitas bahwa langkah yang kurang menguntungkan akan dibuang (*pruned*) menjadi lebih besar. Waktu komputasi yang berkurang dari hasil optimasi dengan *Alpha-Beta Pruning* memungkinkan program untuk meningkatkan kedalaman pencarian, dan dengan begitu diharapkan akan menghasilkan penentuan langkah yang lebih baik.

Bila dibandingkan dengan Reversi dengan *Greedy Engine* yang pernah diimplementasikan sebelumnya, Reversi yang menggunakan kombinasi *Branch and Bound* dan *Greedy by Number* ini jelas lebih baik dan lebih cerdas. Dapat dibuktikan dari hasil pengujian yang menunjukkan bahwa Reversi dengan pencarian menggunakan *Branch and Bound* berkedalaman minimal 3 selalu mengalahkan Reversi dengan *engine Greedy by Number*.

## REFERENSI


[1] Hamed Ahmadi, "An Introduction to Game Tree Algorithms", 2008.

[2] Ir. Rinaldi Munir, M.T., "Diktat Kuliah IF3051 Strategi Algoritma",Program Studi Teknik Informatika, 2009.

[3] I.Y.B. Aditya Eka Prabawa W., Stephen Herlambang & F.B. Dian Paskalis, "Aplikasi Algoritma Greedy pada Permainan Reversi", Program Studi Teknik Informatika, 2009