Analisis Kompleksitas Algoritma Alpha-Beta Pruning pada Mesin Catur

Muhammad Aulia Azka 13523137
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13523137@itb.ac.id, auliaazka2506@gmail.com

***Abstract*—Algoritma *Alpha-Beta Pruning* merupakan salah satu pendekatan atau teknik utama yang banyak dimanfaatkan dalam proses pengambilan keputusan pada permainan dua pemain *(two-players-games)*. Salah satu game tersebut adalah catur. Melalui metode ini, proses pencarian langkah yang paling baik dapat dioptimalkan dengan mengurangi sejumlah jalur (cabang) yang tidak berpengaruh terhadap hasil akhir. Jika algoritma ini dibandingkan dengan algoritma *minimax* murni, *Alpha-Beta Pruning* terbukti mampu menurunkan jumlah simpul yang perlu dievaluasi. Hal ini menyebabkan beban komputasi menjadi lebih efisien. Dalam permainan catur, setiap posisi bidak di atas papan memiliki berbagai kemungkinan gerakan (langkah) yang perlu diperiksa. Melalui makalah ini, analisis menyeluruh mengenai kompleksitas dan kinerja *Alpha-Beta Pruning* dalam mesin catur dilakukan. Pembahasan meliputi landasan teori dan analisis.**

***Kata kunci*—Alpha-Beta Pruning, Kompleksitas, Catur, Minimax.**

I. PENDAHULUAN

Permainan catur adalah salah satu permainan papan yang telah lama menjadi objek penelitian di bidang kecerdasan buatan. Setiap posisi pada papan catur memiliki banyak kemungkinan langkah yang dapat ditempuh oleh kedua pemain, dan masing-masing langkah tersebut membuka ragam konsekuensi yang membutuhkan analisis mendalam. Catur memiliki kompleksitas yang sangat tinggi, serta berbagai kemungkinan langkah yang harus diperhitungkan oleh mesin catur. Salah satu algoritma yang terkenal untuk melakukan pencarian langkah terbaik adalah algoritma minimax. Namun, algoritma tersebut memiliki kelemahan dalam hal jumlah node yang dievaluasi.

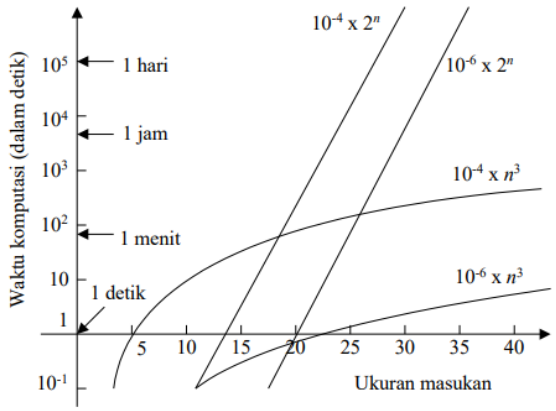
Untuk mengatasi kelemahan tersebut, algoritma *Alpha-Beta Pruning* dibutuhkan. *Alpha-Beta Pruning* melakukan pemangkasan (pruning) pada cabang – cabang yang tidak diperlukan untuk diperhitungkan lebih lanjut. Mekanisme pemangkasan ini menggunakan dua parameter yaitu alpha (α) dan beta (β). Konsep ini sangat penting, mengingat jumlah gerakan rata – rata dalam catur bisa berkisar antara 35 hingga 40, bahkan lebih. Artinya, tanpa upaya pemangkasan, pohon pencarian dapat tumbuh secara eksponensial. Kedua parameter tersebut berfungsi sebagai batas (bound) untuk nilai terbaik yang bisa dicapai. Dengan memanfaatkan kedua parameter tersebut, algoritma dapat menentukan cabang – cabang mana yang tidak akan mempengaruhi hasil akhir secara lebih cepat.

Melalui penerapan *Alpha-Beta Pruning*, mesin catur tidak perlu lagi "membuang waktu" untuk mengulas variasi langkah yang jelas – jelas tidak menjanjikan peningkatan hasil.

II. DASAR TEORI

a. Kompleksitas Algoritma

Sebuah algoritma selain harus menghasilkan solusi yang benar terhadap suatu permasalahan, tetapi juga harus berjalan secara efisien tanpa harus memakan memori yang tidak sedikit dan tidak memakan waktu yang lama.



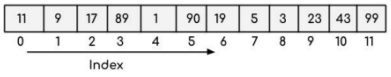
Gambar 1. Diambil dari (https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/25-Kompleksitas-Algoritma-Bagian1-2024.pdf)

Ada dua macam kompleksitas algoritma yaitu kompleksitas waktu dan kompleksitas ruang. Kompleksitas waktu, T(n), diukur dari jumlah tahapan komputasi yang dilakukan di dalam algoritma sebagai fungsi dari ukuran masukkan n. Kedua adalah

kompleksitas ruang, S(n), diukur dari memori yang digunakan oleh struktur data yang terdapat dalam algoritma sebagai fungsi dari ukuran masukan n. Dengan menggunakan besaran kompleksitas waktu/ruang algoritma, laju peningkatan waktu (ruang) yang diperlukan algoritma dengan meningkatnya ukuran masukan n dapat ditentukan.

Lihatlah contoh berikut:

```
sum ←0
for i ← 1 to n do
    sum ← sum + a[i]
endfor
rata_rata ← sum/n
```



Gambar 2. Diambil dari (https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/25-Kompleksitas-Algoritma-Bagian1-2024.pdf)

```
k←1
ketemu ← false
while (k ≤ n) and (not ketemu) do
  if a_k = x then
     ketemu←true
  else
     k ← k + 1
  endif
endwhile
if ketemu then   { x ditemukan }
   idx←k
else
   idx← –1       { x tidak ditemukan }
endif
```

Gambar 3. Diambil dari (https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/25-Kompleksitas-Algoritma-Bagian1-2024.pdf)

Operasi dasar pada algoritma di atas adalah operasi penjumlahan elemen – elemen list (sum <- sum + a[i]) yang dilakukan sebanyak n kali. Maka kompleksitas waktu adalah T(n) = n

Kompleksitas waktu dibagi menjadi tiga macam:

1. Tmax(n) :
   Kompleksitas waktu untuk kasus terburuk.
   Kebutuhan waktu maksimum

2. Tmin(n) :
   Kompleksitas waktu terbaik.
   Kebutuhan waktu minimum.

3. Tavg(n) :
   Kompleksitas waktu rata – rata.
   Kebutuhan waktu secara rata – rata.

Contoh Algoritma Sequential Search

1. Kasus terbaik: a1 = x
   Tmin(n) = 1

2. Kasus terburuk: an = x atau x tidak ditemukan
   Tmax(n) = n

3. Kasus rata – rata: Jika x ditemukan pada posisi ke-j, maka operasi perbandingan (ak = x) akan dijalankan sebajak j kali

$$T_{avg}(n) = \frac{(1+2+3+...+n)}{n} = \frac{\frac{1}{2}n(1+n)}{n} = \frac{(n+1)}{2}$$

Gambar 4. Diambil dari (https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/25-Kompleksitas-Algoritma-Bagian1-2024.pdf)

b. Kompleksitas Waktu Asimptotik

Kinerja algoritma – algoritma pengurutan seperti selection sort dan bubble sort baru akan terlihat ketika mengurutkan list berukuran besar, misal 99999 elemen. Untuk itu dibutuhkannya suatu notasi kompleksitas algoritma yang dapat memperlihatkan kinerja algoritma untuk n yang besar. Notasi kompleksitas algoritma tersebut dinamakan kompleksitas waktu asimptotik.

n	$T(n) = 2n^2 + 6n + 1$	n^2
10	261	100
100	20.601	10.000
1000	2.006.001	1.000.000
10.000	200.060.001	100.000.000

Gambar 5. Diambil dari (https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/26-Kompleksitas-Algoritma-Bagian2-2024.pdf)

c. Notasi O-Besar (Big-O)

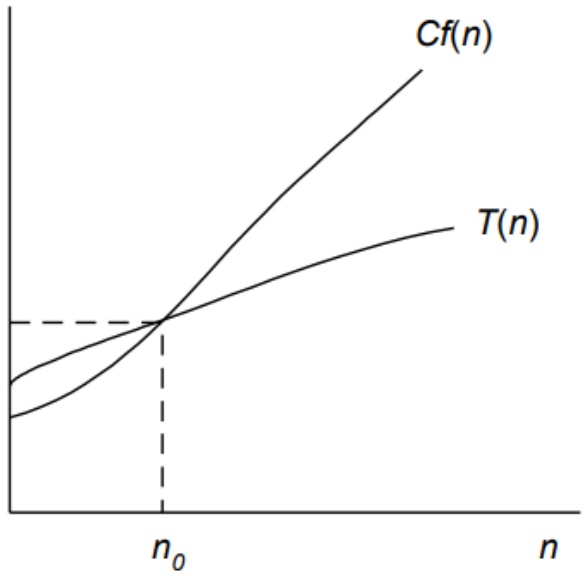
Notasi "O" atau bisa juga disebut notasi "O-Besar" (Big-O) adalah notasi waktu asimptotik. Definisi dari notasi O-Besar adalah:

T(n) = O(f(n)) bila terdapat konstanta C dan n0 sedemikian sehingga

T(n) ≤ C f(n)

Untuk n ≥ n0

f(n) adalah batas lebih atas (upper bound) dari T(n) untuk n yang besar



Gambar 6. Diambil dari (https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/26-Kompleksitas-Algoritma-Bagian2-2024.pdf)

Fungsi f(n) biasanya dipilih dari fungsi – fungsi standard seperti $1, n^2, n^3, \ldots, \log n, n \log n, 2^n, n!$.

Nilai C dan n0 berjumlah tak-berhingga yang memenuhi T(n) ≤ C f(n), cukup menentukan satu pasang saja (C, n0) yang memenuhi definisi sehingga T(n) = O(f(n))

Contoh Tunjukan bahwa $2n^2 + 6n + 1 = O(n^2)$

$2n^2 + 6n + 1 = O(n^2)$ karena
$2n^2 + 6n + 1 \leq 2n^2 + 6n^2 + n^2 = 9n^2$
C = 9, f(n) = n^2, n0 = 1

Teorema 1: Bila $T(n) = a_m n^m + a_{m-1} n^{m-1} + \ldots + a_1 n + a_0$ adalah polinom derajat ≤ m maka $T(n) = O(n^m)$.

Jadi untuk menentukan notasi Big-O, cukup melihat suku yang mempunyai pangkat terbesar di dalam T(n). Contoh:

$$T(n) = 5 = 5n^0 = O(n^0) = O(1)$$
$$T(n) = 2n + 3 = O(n)$$
$$T(n) = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2} = O(n^2)$$
$$T(n) = 3n^3 + 2n^2 + 10 = O(n^3)$$

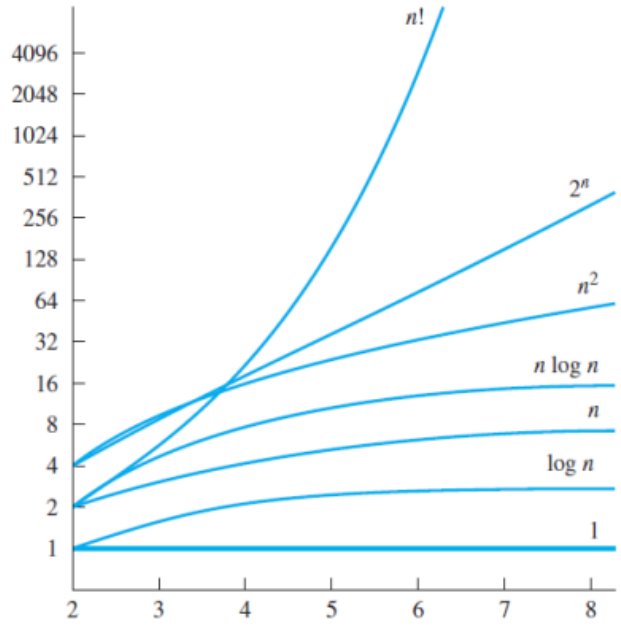
Gambar 7. Diambil dari (https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/26-Kompleksitas-Algoritma-Bagian2-2024.pdf)

Teorema 1 dapat digeneralisir menjadi:

1. Eksponensial mendominasi sembarang perpangkatan
2. Perpangkatan mendominasi ln(n)
3. Semua bentuk logaritma tumbuh pada laju yang sama
4. Bentuk n log n lebih cepat daripada n tetapi lebih lambat daripada n^2

TEOREMA 2. Misalkan $T_1(n) = O(f(n))$ dan $T_2(n) = O(g(n))$, maka
(a) $T_1(n) + T_2(n) = O(f(n)) + O(g(n)) = O(\max(f(n), g(n))$
(b) $T_1(n)T_2(n) = O(f(n))O(g(n)) = O(f(n)g(n))$
(c) $O(cf(n)) = O(f(n))$, c adalah konstanta
(d) $f(n) = O(f(n))$

Gambar 8. Diambil dari (https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/26-Kompleksitas-Algoritma-Bagian2-2024.pdf)



Gambar 9. Diambil dari (https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/26-Kompleksitas-Algoritma-Bagian2-2024.pdf)

Kompleksitas O(1) ini mengindikasikan waktu pelaksanaan algoritma tetap, yang berarti tidak bergantung pada ukuran masukan. Algoritma yang memiliki kompleksitas O(1) adalah algoritma yang memiliki instruksi yang dijalankan sebanyak satu kali (tidak ada pengulangan atau loop).

Kompleksitas O(log n) mengindikasikan kompleksitas waktu logaritmik yang berarti laju pertumbuhan waktunya berjalan lebih lambat daripada pertumbuhan n. Algoritma yang memiliki kompleksitas O(log n) adalah algoritma yang memecahkan persoalan besar dengan mentransformasikannya atau membaginya menjadi beberapa persoalan yang lebih kecil dan berukuran sama. Contoh algoritma tersebut adalah algoritma binary search.

Kompleksitas O(n) mengindikasikan algoritma yang waktu eksekusinya lanjar (linier). Umumnya terdapat pada kasus yang setiap elemen masukannya dikenai proses yang sama. Contoh algoritma yang memiliki kompleksitas O(n) adalah algoritma sequential search.

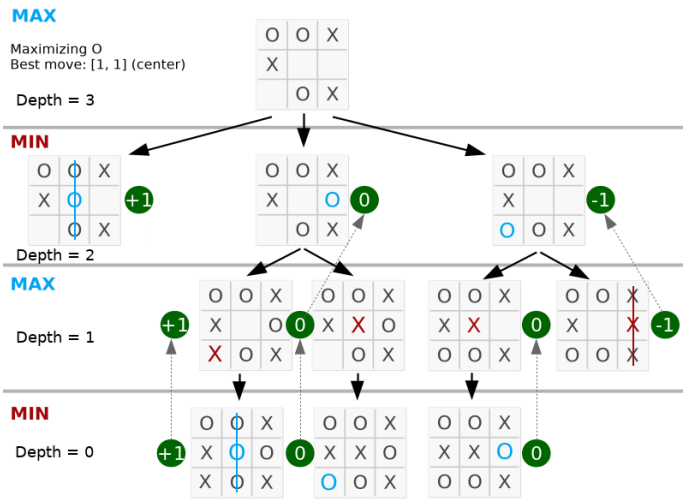
Kompleksitas O(n log n) mengindikasikan algoritma yang memecahkan persoalan menjadi beberapa persoalan yang lebih kecil, menyelesaikan tiap persoalan kecil tersebut secara independen kemudian menggabungkan solusi masing – masing persoalan. Contoh algoritma yang memiliki kompleksitas waktu O(n log n) adalah devide and conquer.

Kompleksitas O(n^2) mengindikasikan algoritma yang memiliki waktu eksekusinya kuadratik. Umumnya algoritma yang termasuk kelompok ini memproses setiap masukan dalam dua buah loop bersarang. Contoh algoritma tersebut adalah bubble sort.

d. Algoritma Minimax

Algoritma minimax adalah suatu algoritma yang berfungsi untuk menentukan langkah optimal dalam suatu permainan dua pemain yang bersifat *zero-sum*, Contoh permainan tersebut adalah catur. Pada permainan catur, algoritma minimax biasanya diimplementasikan sebagai berikut:

1. Membuat pohon pencarian yang merepresentasikan semua kemungkinan langkah dari dua pemain.
2. Mengevaluasi nilai pada posisi daun berdasarkan fungsi evaluasi.
3. Mengalirkan nilai dari level terdalam ke level teratas dengan pergantian antara memilih nilai maksimum dan minimum.
4. Langkah terbaik dipilih berdasarkan nilai akhir di simpul.

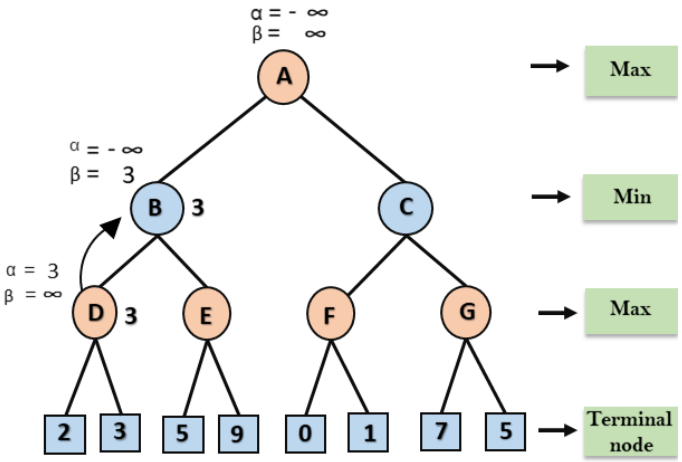


Gambar 10. Ilustrasi algoritma Minimax pada permainan TicTacToe diambil dari (https://viandwi24.medium.com/membuat-ai-tictactoe-dengan-algoritma-minimax-javascript-part-2-dan-mengenal-apa-itu-minimax-908c7f0a9f8c)

e. Alpha-Beta Pruning

Alpha-Beta Pruning adalah teknik optimasi yang memanfaatkan informasi dari 2 parameter yaitu alpha (α) yang merupakan batas bawah dan beta (β) yang merupakan batas atas untuk memangkas (prune) cabang – cabang pohon yang tidak mungkin mempengaruhi hasil akhir.

Proses pemangkasan ini dilakukan ketika suatu cabang memiliki nilai yang lebih buruk dibandingkan alpha dan beta yang berlaku. Pemangkasan tersebut dapat diartikan bahwa cabang tersebut tidak akan mengubah hasil akhir, sehingga tidak perlu dievaluasi.



Gambar 11. Diambil dari (https://www.javatpoint.com/ai-alpha-beta-pruning)

III. ANALISIS

Untuk melakukan analisis diperlukan pemilihan parameter – parameter yang digunakan untuk analisis.

1. *Deph* (Kedalaman Pencarian)
   Pada permainan catur, kedalaman menggambarkan seberapa jauh langkah ke depan yang akan dianalisis oleh algoritma. Pada analisis ini digunakan kedalaman = 4 dan 6.
2. *Branching Factor* (BF)
   Branching Factor merupakan banyaknya jalur yang bisa diambil di setiap langkah. Pada analisis ini dipilih BF = 10 dan 15
3. Fungsi Evaluasi
   Fungsi Evaluasi berfungsi untuk memberikan nilai pada posisi tertentu di permainan, terutama di simpul daun ketika sudah mencapai kedalaman maksimum atau kondisi terminal.
4. *Move Ordering*
   Move Ordering adalah urutan langkah yang akan diperiksa oleh algoritma. Urutan langkah yang "bagus" (*heuristik)* bertujuan untuk memprioritaskan langkah – langkah yang paling bagus terlebih dahulu. Sedangkan urutan "acak" (*random)* tidak akan menyusun langkah – langkah berdasarkan prioritas.

Selain itu pada analisis ini penulis akan membandingkan dua Algoritma yaitu *minimax* tanpa *Alpha-Beta.* Pada bagian ini, semua cabang pohon pencarian akan ditelusuri tanpa ada pemangkasan.

```python
def minimax_no_pruning(node, is_maximizing_player, counter):
    """
    node: simpul yang sedang diolah
    is_maximizing_player: boolean True/False
    counter: dictionary atau list untuk menghitung node yang dievaluasi
    """
    # Hitung setiap kunjungan node
    counter["visited"] += 1

    # Jika leaf, kembalikan value
    if node.is_leaf():
        return node.value

    if is_maximizing_player:
        best_val = float('-inf')
        for child in node.children:
            val = minimax_no_pruning(child, False, counter)
            best_val = max(best_val, val)
        return best_val
    else:
        best_val = float('inf')
        for child in node.children:
            val = minimax_no_pruning(child, True, counter)
            best_val = min(best_val, val)
        return best_val
```

Gambar 12. Algoritma minimax tanpa alpha-beta

Algoritma kedua adalah algoritma *minimax* dengan *Alpha-Beta*. Pada bagian ini akan dijalankan algoritma minimax, tetapi pada bagian ini setiap kali mendapatkan suatu simpul, nilainya akan dibandingkan dengan alpha dan beta. Jika nilai yang didapat tidak akan mengubah hasil di level atas, evaluasi pada sisa anak di cabang itu akan dihentikan. Hasil akhirnya akan tetap sama dengan minimax biasa, tetapi jumlah simpul yang diperiksa tentu akan lebih sedikit dibandingkan dengan minimax biasa.

```python
def minimax_alpha_beta(node, depth, alpha, beta, is_maximizing_player, counter):
    counter["visited"] += 1

    if node.is_leaf():
        return node.value

    if is_maximizing_player:
        value = float('-inf')
        for child in node.children:
            value = max(value, minimax_alpha_beta(child, depth+1, alpha, beta, False, counter))
            alpha = max(alpha, value)
            if beta <= alpha:
                break  # pruning
        return value
    else:
        value = float('inf')
        for child in node.children:
            value = min(value, minimax_alpha_beta(child, depth+1, alpha, beta, True, counter))
            beta = min(beta, value)
            if beta <= alpha:
                break  # pruning
        return value
```

Gambar 13. Algoritma minimax dengan alpha-beta

Setelah menjalankan kode didapatkan hasil berupa tabel dengan parameter Deph, BF, Ordering, V_NoPrunse, V_AlphaBeta, Val_Mini, dan Val_AB.

Deph	BF	Ordering	V_NoPrune	V_AlphaBeta	Val_Mini	Val_AB
4	10	Random	111111	2308	-69	-69
4	10	heuristic	111111	1987	-51	-51
4	15	random	54241	6244	-71	-71
4	15	heuristic	54241	8766	-76	-76
6	10	random	11111111	66458	-66	-66
6	10	heuristic	11111111	60859	-66	-66
6	15	random	12204241	265727	-74	-74
6	15	heuristic	12204241	325994	-74	-74

Keterangan:
Deph: Kedalaman (4 atau 6)
BF: Branching Factor (10 atau 15)
Ordering: Metode pengurutan langkah
V_NoPrune: Jumlah node yang dievaluasi oleh minimax tanpa alpha-beta
V_AlphaBeta: Jumlah node yang dievaluasi oleh minimax dengan alpha-beta
Val_Mini dan Val_AB: Skoe akhir yang dihasilkan, seharusnya sama karena alpha-beta tidak mengubah hasil akhir (hanya memotong jalur yang tidak relevan)

Pada kolom V_NoPrunde dan V_AlphaBeta, terlihat perbedaan jumlah node yang sangat besar jika dibandingkan dengan Dephnya (4 dan 6). Hal ini mengindikasi semakin dalam dan semakin besar branching factor, minimax tanpa pemangkasan akan mengecek pohon secara eksponensial. Hal ini menyebabkan jumlah node yang dievaluasi melonjak drastis. Sementara itu, alpha-beta mampu memangkas cabang – cabang yang tidak akan mempengaruhi hasil akhir, sehingga jumlah node yang diperiksa lebih sedikit.

Jika ditinjau pengaruh branching factor maka didapatkan BF = 15 menghasilkan node total yang lebih kecil atau lebih besar tergantung dengan struktur pohon yang acak. Seperti contoh pada Deph = 4, BF = 15 justru memiliki jumlah node yang lebih rendah (54.241) dibandingkan dengan BF = 10 (111.111). Hal ini dapat disebakan karena struktur pohon yang acak.

Jika ditinjau pengaruh dari Val_Mini dan Val_AB, hasilnya selalu sama (Val_Mini = -69 dan Val_AB = -69), hal ini menegaskan bahwa *alpha-beta pruning* tidak akan mengubah hasil akhir. Tujuan Alpha-Beta sendiri hanya untuk memangkas jalur.

IV. KESIMPULAN

Berdasarkan proses analisis yang telah dilakukan terhadap penerapan *Alpha-Beta Pruning* dalam algoritma minimax pada permainan catur, dapat disimpulkan metode ini dapat memangkas jumlah node yang harus dievaluasi cukup signifikan. Strategi *move ordering* cukup memiliki peranan penting dalam meningkatkan efektivitas pemangkasan, sehingga waktu komputasi dan jumlah simpul yang ditelusuri dapat dikurangi. Di sisi lain, *Alpha-Beta Pruning* tidak mengubah hasil akhir dari nilai minimax. Hal ini mengindikasi bahwa penerapan *alpha-beta pruning* dapat menurunkan kompleksitas waktu algoritma tanpa mengubah hasil akhir.

V. LAMPIRAN

Source code:
https://github.com/Azzkaaaa/Analisis-Kompleksitas-Algoritma-Alpha-Beta-Pruning-pada-Mesin-Catur

VI. UCAPAN TERIMA KASIH

Terima kasih kepada Tuhan yang Maha Esa karena atas karunia dan berkatnya penulis dapat menyelesaikan makalah ini dengan cukup baik. Tak lupa penulis ucapkan terima kasih kepada keluarga dan teman – teman karena telah memberikan dukungan dan semangat kepada penulis sehingga penulis dapat menyelesaikan masalah ini. Terima kasih juga kepada Dr. Nur Ulfa Maulidevi, S.T, M.Sc. dan Pak Arrival Dwi Sentosa, S. Kom., M.T. selaku dosen pengampu mata kuliah matematika diskrit karena telah memberikan ilmu dan pengetahuan sehingga penulis dapat menerapkannya untuk menulis makalah ini.

REFRENSI

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/25-Kompleksitas-Algoritma-Bagian1-2024.pdf. Diakses pada 28 Desember 2024
[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/26-Kompleksitas-Algoritma-Bagian2-2024.pdf. Diakses pada 29 Desember 2024
[3] https://www.javatpoint.com/ai-alpha-beta-pruning. Diakses pada 30 Desember 2024
[4] https://viandwi24.medium.com/membuat-ai-tictactoe-dengan-algoritma-minimax-javascript-part-2-dan-mengenal-apa-itu-minimax-908c7f0a9f8c. Diakses pada 30 desember 2024

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 26 Desember 2024

Muhammad Aulia Azka 13523137