Perbandingan Pendekatan DFS dan BFS Berbasis Evaluasi untuk Pemilihan Langkah Optimal dalam Catur

Nathanael Rachmat - 13523142
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: nathanael.rachmat@gmail.com , 13523142@std.stei.itb.ac.id

Abstrak*—Permainan catur merupakan salah satu domain paling menantang dalam kecerdasan buatan, di mana penentuan langkah optimal memerlukan eksplorasi ruang keadaan yang sangat besar. Makalah ini menyajikan perbandingan dua pendekatan algoritma pencarian, yaitu Depth-First Search (DFS) dengan pemangkasan (pruning) dan Best-First Search (sebuah varian dari Breadth-First Search yang diimplementasikan dengan antrian prioritas), untuk membangun pohon permainan dan memilih langkah terbaik dalam sebuah mesin catur. Pendekatan DFS yang diimplementasikan akan menelusuri permainan secara mendalam dan memangkas cabang pencarian yang mengarah pada kerugian besar secara evaluasi. Di sisi lain, Best-First Search secara selektif mengeksplorasi langkah-langkah yang paling menjanjikan terlebih dahulu berdasarkan skor evaluasi tertinggi. Implementasi kedua algoritma ini dalam program simulasi catur menjadi landasan untuk analisis komparatif. Kontribusi utama dari tulisan ini adalah menyajikan hasil eksperimen dari kedua metode tersebut, membandingkan keefektifannya dalam hal kualitas langkah yang dipilih dan sumber daya komputasi yang digunakan, sehingga memberikan wawasan mengenai karakteristik dan trade-off dari masing-masing strategi dalam konteks pencarian heuristik.

Kata Kunci — Catur, Strategi Algoritma, Depth-First Search (DFS), Breadth-First Search (BFS), Best-First Search, Branch and Bound, Pencarian Ruang Keadaan, Fungsi Evaluasi.

I. PENDAHULUAN

Catur adalah permainan strategi klasik yang telah lama menjadi tolok ukur dalam pengembangan kecerdasan buatan. Kompleksitasnya yang luar biasa, dengan jumlah kemungkinan posisi legal yang diperkirakan mencapai 10^{120} (dikenal sebagai Shannon Number), menjadikan catur sebagai domain yang ideal untuk menguji batas kemampuan komputasi. Representasi permainan catur sebagai pohon pencarian, di mana setiap simpul adalah posisi dan setiap cabang adalah langkah, merupakan pendekatan fundamental dalam informatika untuk memodelkan dan menyelesaikan masalah ini.

Pada penelitian yang penulis lakukan sebelumnya untuk mata kuliah Matematika Diskrit, telah diimplementasikan sebuah pendekatan untuk menemukan langkah catur dengan memanfaatkan konsep pohon permainan dan analisis kombinatorial. Pendekatan tersebut, yang pada dasarnya membangkitkan dan mengevaluasi seluruh kemungkinan langkah, tanpa disadari saat itu merupakan bentuk murni dari strategi brute force. Hal ini terjadi sebelum penulis mendalami secara formal berbagai paradigma dalam mata kuliah Strategi Algoritma. Meskipun pendekatan brute force tersebut menjamin penemuan solusi, ia memiliki kelemahan fatal dalam praktiknya. Akibat ledakan kombinatorial, proses pencarian menjadi sangat tidak efisien dan memakan waktu yang sangat lama, bahkan untuk kedalaman pohon yang relatif dangkal. Keterbatasan ini menjadikan algoritma brute force murni tidak praktis untuk aplikasi catur yang menuntut pengambilan keputusan dalam waktu yang wajar.

Menyadari kelemahan tersebut, studi dalam mata kuliah Strategi Algoritma menawarkan berbagai paradigma yang lebih canggih untuk mengatasi masalah inefisiensi. Makalah ini berfokus pada implementasi dan perbandingan dua strategi optimasi untuk menggantikan pendekatan brute force. Strategi pertama adalah Depth-First Search (DFS) yang dimodifikasi dengan teknik pemangkasan (pruning). Terinspirasi dari konsep algoritma runut-balik (backtracking), pendekatan ini akan menghentikan penelusuran sebuah cabang pencarian secara prematur jika terdeteksi bahwa cabang tersebut tidak akan menghasilkan solusi yang lebih baik dari yang telah ditemukan.

Strategi kedua adalah implementasi Best-First Search, sebuah varian dari Breadth-First Search (BFS) yang dipengaruhi oleh prinsip algoritma Greedy dan Branch and Bound. Berbeda dengan BFS standar yang menjelajah level per level, Best-First Search menggunakan fungsi evaluasi heuristik untuk memprioritaskan simpul (langkah) yang paling menjanjikan untuk dieksplorasi terlebih dahulu. Dengan cara ini, pencarian diarahkan ke wilayah yang paling potensial dalam pohon permainan, sehingga diharapkan dapat menemukan langkah optimal dengan lebih cepat.

Oleh karena itu, penelitian ini bertujuan untuk mengimplementasikan, menganalisis, dan membandingkan secara langsung efektivitas antara pendekatan DFS dengan pemangkasan dan Best-First Search. Kontribusi makalah ini adalah menyajikan data eksperimental yang menunjukkan bagaimana kedua strategi ini secara signifikan mengoptimalkan proses pencarian langkah catur dibandingkan dengan metode brute force, serta menelaah pertukaran (trade-off) antara

kecepatan komputasi dan kualitas langkah yang dihasilkan oleh masing-masing algoritma.

II. Dasar Teori

A. Permainan Catur dan Pohon Permainan

Permainan catur adalah domain klasik untuk pengujian dan penerapan strategi algoritma. Untuk memahami bagaimana algoritma dapat menentukan langkah optimal, penting untuk terlebih dahulu memahami struktur permainan itu sendiri dan bagaimana struktur tersebut dapat direpresentasikan secara komputasi.

Catur merupakan permainan papan strategi untuk dua pemain yang dimainkan di atas papan 8x8 kotak. Setiap pemain memulai dengan 16 buah catur, yang terdiri dari satu Raja, satu Ratu, dua Benteng, dua Gajah, dua Kuda, dan delapan Pion. Setiap jenis buah memiliki aturan gerak yang unik, dan tujuan akhir dari permainan ini adalah untuk mencapai posisi skakmat (Inggris: checkmate), yaitu sebuah kondisi di mana Raja lawan tidak dapat menghindari serangan [1]. Kemenangan dalam catur menuntut kemampuan berpikir strategis, taktis, dan perhitungan langkah yang cermat.

Meskipun aturan dasarnya relatif sederhana, catur memiliki tingkat kompleksitas yang sangat tinggi. Tantangan utama dalam catur dari sudut pandang komputasi adalah ledakan kombinatorial, yaitu pertumbuhan jumlah kemungkinan posisi yang sangat cepat seiring dengan bertambahnya jumlah langkah. Diperkirakan terdapat sekitar 10^{43} posisi legal yang mungkin terjadi di papan catur, dan jumlah total kemungkinan permainan unik diestimasi oleh Claude Shannon mencapai 10^{120}, sebuah angka yang dikenal sebagai Angka Shannon [2]. Skala yang luar biasa besar ini membuat analisis yang menyeluruh terhadap semua kemungkinan langkah menjadi tidak praktis, bahkan bagi komputer paling kuat sekalipun. Oleh karena itu, diperlukan pendekatan yang lebih cerdas daripada sekadar memeriksa setiap kemungkinan satu per satu.

Dalam ilmu komputer, cara standar untuk merepresentasikan permainan seperti catur adalah dengan menggunakan struktur data pohon permainan (game tree). Pohon permainan adalah sebuah model matematis yang memvisualisasikan seluruh kemungkinan alur dari sebuah permainan. Dalam konteks catur, komponen-komponen pohon ini didefinisikan sebagai berikut:

- Akar (Root): Merepresentasikan posisi awal atau posisi saat ini di papan catur yang sedang dianalisis.
- Simpul (Node): Setiap simpul dalam pohon merepresentasikan sebuah posisi papan yang unik dan legal.
- Cabang (Branch atau Edge): Setiap cabang yang menghubungkan dua simpul merepresentasikan sebuah langkah catur yang valid, yang mengubah posisi dari simpul induk (parent) ke simpul anak (child).
- Daun (Leaf): Simpul-simpul di ujung pohon yang tidak memiliki anak. Sebuah simpul daun dapat merepresentasikan kondisi akhir permainan (seperti skakmat atau remis) atau titik di mana analisis berhenti karena telah mencapai batas kedalaman yang ditentukan.
- Kedalaman (Depth): Ukuran yang menunjukkan seberapa banyak langkah ke depan (dikenal juga sebagai ply) yang dianalisis oleh algoritma dari posisi saat ini.

Dengan model pohon ini, masalah penentuan langkah catur dapat ditransformasikan menjadi masalah pencarian jalur (path finding) di dalam sebuah graf.

Bagi sebuah mesin catur, proses "berpikir" adalah proses membangun dan menelusuri sebagian dari pohon permainan raksasa ini, dimulai dari simpul akar. Karena pohon permainan catur secara keseluruhan terlalu besar untuk dieksplorasi sepenuhnya, algoritma harus secara selektif menjelajahi cabang-cabang yang paling menjanjikan. Algoritma akan mencari sebuah jalur dari akar ke salah satu simpul daun yang menghasilkan posisi dengan nilai evaluasi paling menguntungkan bagi pemain. Dengan demikian, pemilihan strategi algoritma pencarian menjadi faktor krusial yang menentukan kecepatan dan kualitas keputusan sebuah mesin catur.

B. Fungsi Evaluasi Posisi

Inti dari kecerdasan sebuah mesin catur terletak pada fungsi evaluasi heuristik. Fungsi ini bertugas memberikan skor numerik pada setiap simpul di pohon permainan. Skor ini merupakan estimasi tentang seberapa menguntungkan posisi tersebut bagi salah satu pemain. Secara konvensi, skor positif menandakan keunggulan bagi pemain Putih, sementara skor negatif menandakan keunggulan bagi pemain Hitam. Skor mendekati nol mengindikasikan posisi yang seimbang.

Fungsi evaluasi yang baik akan mempertimbangkan berbagai faktor strategis dan taktis. Dalam implementasi proyek ini, fungsi evaluasi yang digunakan terinspirasi dari prinsip-prinsip yang juga digunakan oleh mesin catur modern seperti Stockfish [3], yang mencakup:

- Materi (Material): Komponen paling dasar, yaitu nilai total dari buah catur yang dimiliki setiap pemain di papan.
- Mobilitas (Mobility): Jumlah langkah legal yang tersedia. Semakin banyak pilihan langkah yang dimiliki, umumnya posisi dianggap semakin baik.
- Keamanan Raja (King Safety): Mengukur seberapa aman posisi Raja dari ancaman dan serangan lawan.
- Struktur Pion (Pawn Structure): Menilai kekuatan dan kelemahan formasi pion, seperti pion terisolasi, pion tumpuk, atau pion bebas.
- Kontrol Ruang (Space Control): Mengukur penguasaan atas area-area penting di papan catur.

Dalam konteks makalah ini, skor dari fungsi evaluasi ini menjadi penentu utama. Skor inilah yang digunakan sebagai kriteria untuk melakukan pruning pada algoritma DFS dan menjadi dasar prioritas dalam antrian pada algoritma Best-First Search.

C. *Algoritma Brute Force sebagai Garis Dasar*

Pendekatan Brute Force adalah strategi paling mendasar untuk menelusuri pohon permainan. Cara kerjanya adalah dengan membangkitkan dan memeriksa setiap kemungkinan langkah secara sistematis hingga kedalaman tertentu. Untuk setiap langkah, ia akan membangkitkan semua balasan lawan, kemudian semua balasan baliknya, dan seterusnya, hingga mencapai batas kedalaman yang ditentukan.

Meskipun pendekatan ini menjamin kelengkapan, yang berarti ia pasti akan menemukan jalur permainan jika ada dalam jangkauan pencariannya. Kelemahan utamanya adalah inefisiensi. Akibat sifat ledakan kombinatorial dalam catur, jumlah simpul yang harus diperiksa tumbuh secara eksponensial. Hal ini membuat Brute Force murni sangat lambat dan memakan sumber daya komputasi yang besar, sehingga tidak praktis untuk analisis dengan kedalaman yang signifikan. Lebih jauh, tanpa mekanisme pemangkasan yang cerdas, evaluasi akhir dapat terdistorsi oleh satu cabang permainan yang di dalamnya terdapat blunder fatal dari lawan, yang secara artifisial menaikkan skor sebuah langkah yang sebenarnya tidak bagus. Keterbatasan inilah yang menjadi motivasi utama untuk mencari strategi algoritma yang lebih optimal.

D. *Pencarian Mendalam: DFS with Pruning*

Salah satu solusi untuk mengatasi kelemahan Brute Force adalah dengan menggunakan Depth-First Search (DFS) yang dikombinasikan dengan teknik *pruning*. DFS adalah strategi penelusuran yang menjelajahi sebuah cabang pohon permainan sedalam mungkin sebelum melakukan *backtracking* untuk mencoba cabang lain.

Pendekatan ini menjadi jauh lebih efisien dengan adanya pemangkasan. Pruning adalah teknik memotong atau mengabaikan cabang-cabang dari pohon pencarian yang dianggap tidak akan menghasilkan solusi yang lebih baik. Dalam implementasi yang dilakukan, pemangkasan bersifat heuristik: jika sebuah langkah mengarah pada posisi dengan skor evaluasi yang turun secara drastis, maka seluruh cabang yang berasal dari langkah tersebut akan diabaikan.

Dengan cara ini, DFS with Pruning menjawab persoalan Brute Force dengan tidak membuang waktu untuk menganalisis konsekuensi dari langkah-langkah yang jelas-jelas merupakan blunder. Algoritma menjadi lebih fokus pada jalur-jalur permainan yang masuk akal, sehingga secara signifikan mengurangi jumlah simpul yang perlu dieksplorasi dan mempercepat waktu pencarian.

E. *Pencarian Heuristik: Best-First Search dan Branch and Bound*

Pendekatan kedua untuk menyelesaikan masalah inefisiensi Brute Force adalah dengan menggunakan strategi pencarian heuristik yang terinspirasi dari Greedy Best-First Search dan Branch and Bound. Berbeda dengan DFS yang menjelajah secara mendalam, pendekatan ini menjelajah secara lebih terarah.

Algoritma ini menggunakan *priority queue* untuk mengelola simpul-simpul yang akan dieksplorasi. Alih-alih menjelajahi level per level seperti BFS standar, algoritma ini selalu memilih simpul dengan skor evaluasi tertinggi dari dalam antrian untuk dieksplorasi selanjutnya. Sifat *greedy* inilah yang memandu pencarian untuk selalu fokus pada cabang yang paling menjanjikan saat itu.

Implementasi ini pada dasarnya adalah bentuk dari algoritma Branch and Bound. Algoritma tidak hanya memilih jalur terbaik secara lokal, tetapi juga secara implisit menetapkan *bound* pada kualitas solusi. Dengan memprioritaskan simpul-simpul bernilai tinggi, ia dapat menemukan solusi yang baik dengan cepat.

Perbedaan fundamental dalam cara penjelajahan pohon DFS yang vertikal dan dalam, dengan Best-First Search yang horizontal namun terarah dan menghasilkan karakteristik performa yang berbeda. Perbandingan antara kedua pendekatan inilah yang menjadi inti dari analisis dalam makalah ini untuk melihat trade-off antara kecepatan, efisiensi memori, dan kualitas langkah yang dihasilkan.

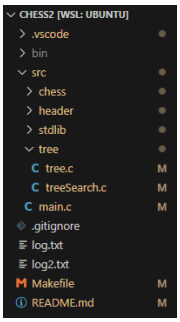
III. Metodologi Penelitian

Bagian ini menguraikan langkah-langkah sistematis yang dilakukan untuk mengimplementasikan, menguji, dan membandingkan kedua strategi algoritma pencarian. Metodologi ini dirancang untuk memastikan bahwa perbandingan kinerja antara DFS with Pruning dan Best-First Search dilakukan secara objektif dan terukur, sehingga kontribusi utama dari makalah ini, yaitu analisis komparatif dari kedua pendekatan, dapat disajikan dengan data yang valid.

A. *Lingkungan dan Perancangan Program*

Penelitian ini diwujudkan melalui sebuah program simulasi catur yang dikembangkan menggunakan bahasa pemrograman C. Program ini dikompilasi dan dijalankan dalam lingkungan Windows Subsystem for Linux (WSL) dengan Ubuntu, serta menggunakan Makefile untuk manajemen kompilasi. Basis kode awal dari program ini diadaptasi dari proyek penulis sebelumnya [4], yang kemudian direstrukturisasi dan dioptimalkan untuk kebutuhan penelitian ini.

Struktur direktori proyek diatur secara modular untuk memisahkan logika-logika yang berbeda, seperti yang terlihat pada Gambar 1.



Gambar 1. Struktur Direktori Program Simulasi Catur

Komponen-komponen utama dalam perancangan program adalah sebagai berikut:

- Direktori src: Berisi seluruh kode sumber yang dibagi lagi menjadi:

    - chess: Mengelola semua logika yang terkait dengan permainan catur, seperti aturan gerak, validasi langkah, dan fungsi evaluasi.
    - tree: Mengandung logika untuk pembuatan dan manipulasi struktur data pohon permainan.
    - main.c: Titik masuk utama program yang berisi game loop dan interaksi dengan pengguna.
- Representasi Papan Catur: Setiap posisi di papan direpresentasikan oleh sebuah struktur data ChessPosition, yang menyimpan informasi tentang letak semua bidak, giliran pemain, dan data relevan lainnya.
- Fungsi Evaluasi Posisi: Fungsi main_evaluation() digunakan untuk menghitung skor heuristik dari setiap ChessPosition. Fungsi ini merupakan hasil penyempurnaan dari implementasi sebelumnya.
- Generator Langkah: Fungsi combination() berperan sebagai generator langkah, yang bertugas untuk membangkitkan semua kemungkinan langkah yang legal dari sebuah posisi tertentu.

B. Implementasi Algoritma Pencarian

Dua algoritma pencarian utama diimplementasikan dalam modul tree untuk membangun pohon permainan dari sebuah posisi secara heuristik.

Pendekatan pertama adalah *Depth-First Search* (DFS) dengan Pemangkasan. Implementasi ini dilakukan dalam fungsi dfsPruned(). Algoritma ini bekerja secara rekursif. Dari sebuah simpul, ia akan memilih satu langkah, membangkitkan simpul anak, lalu memanggil dirinya sendiri untuk simpul anak tersebut hingga mencapai batas kedalaman yang ditentukan. Proses ini dioptimalkan dengan mekanisme pemangkasan: setelah sebuah langkah dievaluasi, perubahan skor akan diperiksa. Jika perubahan skor tersebut melampaui ambang batas EVAL_DROP_THRESHOLD yang menandakan sebuah blunder, maka seluruh cabang dari langkah tersebut akan diabaikan dan algoritma akan melakukan backtrack untuk mencoba langkah lain.

Pendekatan yang kedua adalah *Best-First Search* berbasis *Branch and Bound*. Pendekatan ini diimplementasikan dalam fungsi bbBfs(). Algoritma ini menggunakan sebuah priority queue untuk menyimpan simpul-simpul yang akan dieksplorasi. Pada setiap iterasi, algoritma akan mengambil (pop) simpul dengan skor evaluasi tertinggi dari antrian. Kemudian, semua langkah legal dari simpul tersebut akan dibangkitkan. Simpul-simpul anak yang dihasilkan, setelah melalui filter pemangkasan blunder yang serupa dengan DFS, akan dimasukkan (push) kembali ke dalam antrian prioritas. Proses ini terus berlanjut hingga batas kedalaman (depthLimit) atau batas jumlah simpul (nodeCap) tercapai.

C. Skenario dan Metrik Pengujian

Untuk membandingkan kinerja kedua algoritma secara adil, serangkaian pengujian dilakukan dengan parameter dan metrik yang terdefinisi dengan jelas.

Penelitian ini menggunakan tiga metrik utama untuk mengevaluasi performa kedua algoritma pada setiap giliran mesin:

- Waktu Komputasi: Diukur dalam milidetik (ms), metrik ini menunjukkan seberapa cepat algoritma dapat menentukan langkah terbaik. Data ini diambil dari log Search time.
- Efisiensi Pencarian (Jumlah Simpul): Diukur dari total simpul pohon permainan yang dieksplorasi (Nodes explored). Metrik ini mengukur seberapa efisien algoritma dalam menjelajahi ruang keadaan. Semakin sedikit simpul yang dieksplorasi untuk hasil yang setara atau lebih baik, semakin efisien algoritmanya.
- Kualitas Langkah: Langkah yang dipilih oleh algoritma (Suggested move) akan dianalisis secara kualitatif, salah satunya dengan membandingkannya dengan rekomendasi dari mesin catur yang lebih kuat seperti yang tersedia di Chess.com [5].

Pengujian dilakukan dengan mensimulasikan dua permainan catur lengkap melawan bot dari Chess.com dengan rating ELO 1600. Penulis memainkan langkah yang disarankan oleh masing-masing algoritma sebagai pemain Putih. Setiap algoritma diuji dalam satu permainan penuh, menghasilkan dua skenario permainan yang berbeda untuk dianalisis.

Parameter pengujian untuk kedua algoritma diseragamkan untuk memastikan perbandingan yang adil:

- Kedalaman Pencarian (Depth): Dibatasi hingga kedalaman 3 (ply).
- Batas Maksimum Simpul (khusus BB-BFS): Batas maksimum simpul yang dieksplorasi diatur menjadi 2500.
- Ambang Batas Pemangkasan (Eval Drop Threshold): Ditetapkan pada nilai 10 untuk mendorong pemangkasan yang agresif dan mengurangi waktu tunggu.

Dengan skenario ini, analisis akan fokus pada perbandingan performa kedua algoritma dalam menghadapi lawan yang sama (bot ELO 1600) berdasarkan metrik yang telah ditetapkan di sepanjang permainan..

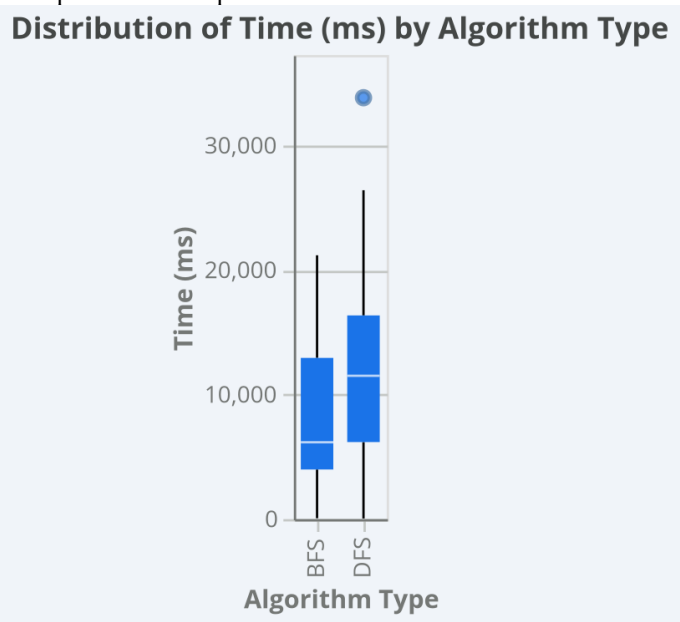
IV. HASIL PENGUJIAN DAN ANALISIS

Pada bab ini, disajikan hasil dari pengujian kedua algoritma yang telah diimplementasikan, yaitu DFS dengan Pemangkasan dan BB-BFS. Pengambilan data dilakukan melalui simulasi permainan melawan bot catur dengan ELO 1600 di Chess.com, di mana setiap giliran mesin dicatat metrik kinerjanya. Log lengkap dari kedua permainan tersebut, log.txt untuk DFS Pruned dan log2.txt untuk BB-BFS, tersedia dalam lampiran

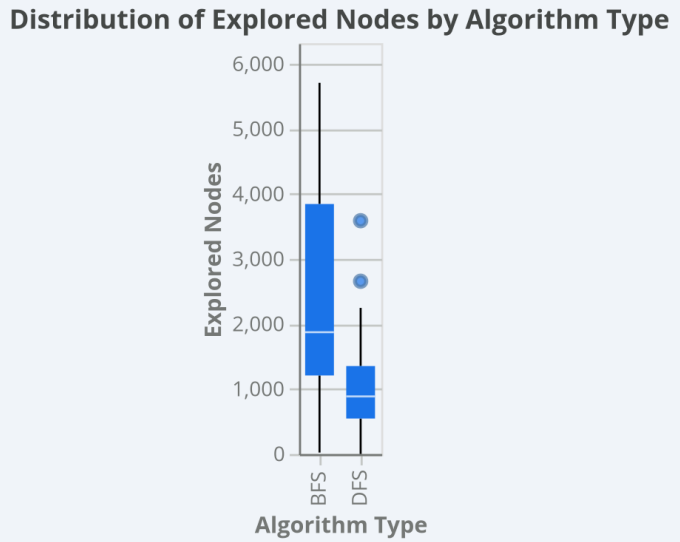
repositori proyek pada GitHub [4] untuk verifikasi dan analisis lebih lanjut.

A. Analisis Visual Distribusi Data

Visualisasi data performa kedua algoritma disajikan dalam bentuk diagram *box plot* untuk memberikan gambaran statistik mengenai distribusi waktu pencarian dan jumlah simpul yang dieksplorasi selama permainan.



Gambar 2. Distribusi waktu hasil pengambilan data



Gambar 3. Distribusi jumlah eksplorasi *node* hasil pengambilan data

Diagram box plot untuk waktu pencarian menunjukkan perbedaan karakteristik yang jelas.

DFS (kanan): Memiliki rentang (jangkauan antar kuartil) yang lebih lebar dan median (garis tengah) yang lebih tinggi, mengindikasikan bahwa secara umum DFS membutuhkan waktu lebih lama untuk membuat keputusan. Adanya outlier (titik di atas) menyoroti kelemahan utama DFS: ia dapat menghabiskan waktu yang sangat lama pada beberapa giliran tertentu, kemungkinan saat terjebak menganalisis cabang pohon yang panjang dan kompleks.

BFS (kiri): Menunjukkan distribusi waktu yang lebih sempit dan median yang lebih rendah. Ini berarti kinerja waktu BFS lebih konsisten dan cenderung lebih cepat daripada DFS, meskipun masih memiliki variabilitas yang cukup besar seperti yang ditunjukkan oleh panjang "kumis" (whisker).

Analisis Distribusi Simpul (Gambar 3): Diagram ini menampilkan trade-off yang terjadi.

DFS (kanan): Sangat efisien dalam hal jumlah simpul yang dieksplorasi. Kotaknya sangat pendek dan terletak di bagian bawah grafik, menunjukkan bahwa pada sebagian besar giliran, DFS hanya memeriksa sedikit simpul. Ini membuktikan bahwa mekanisme pemangkasannya efektif dalam membuang cabang yang tidak relevan.

BFS (kiri): Sebaliknya, BFS secara konsisten menjelajahi jumlah simpul yang jauh lebih besar. Kotaknya yang lebih tinggi dan lebih lebar menunjukkan bahwa BFS melakukan pencarian yang lebih luas dan "boros" secara komputasi untuk menemukan langkah terbaik di antara banyak kandidat.

Secara ringkas, hasil ini menunjukkan pertukaran klasik dalam strategi pencarian: DFS lebih hemat dalam eksplorasi tetapi rentan terhadap penundaan ekstrem, sedangkan BFS lebih boros dalam eksplorasi tetapi memberikan waktu respons yang secara umum lebih cepat dan lebih dapat diprediksi.

B. Analisis Tren dan Pola Permainan

Analisis lebih mendalam terhadap data mentah dari log permainan mengungkapkan beberapa tren dan pola yang menarik:

Memburuknya Kondisi Papan dan Penurunan Skor Evaluasi Seiring Waktu: Seiring berjalannya permainan, terlihat tren umum di mana Evaluasi Awal posisi sebelum giliran mesin cenderung menurun dan menjadi semakin negatif. Hal ini mengindikasikan bahwa posisi mesin secara bertahap memburuk saat melawan bot lawan. Tren ini juga tercermin pada Leaf-sum Pilihan, yang juga cenderung menjadi lebih negatif pada giliran-giliran akhir, menandakan bahwa prospek hasil akhir dari langkah yang dipilih semakin tidak menguntungkan.

Upaya Komputasi yang Variabel dan Hubungannya dengan Kompleksitas Posisi: Baik metrik Simpul Dieksplorasi maupun Waktu (ms) menunjukkan fluktuasi yang signifikan antar giliran, alih-alih sebuah tren linier. Hal ini menyiratkan bahwa beban komputasi untuk setiap langkah sangat bergantung pada kompleksitas posisi pada saat itu. Terdapat korelasi positif yang cukup kuat antara jumlah simpul yang dieksplorasi dan waktu yang dibutuhkan, yang mengonfirmasi bahwa giliran yang memerlukan analisis lebih banyak simpul secara umum akan memakan waktu lebih lama.

Karakteristik Kinerja yang Berbeda Antar Algoritma:

BB-BFS (Best-First Search): Secara rata-rata, algoritma ini menjelajahi jumlah simpul yang jauh lebih banyak dibandingkan DFS. Namun, meskipun melakukan lebih banyak eksplorasi, BB-BFS justru cenderung membutuhkan waktu yang lebih

sedikit per giliran. Distribusi waktunya yang lebih rapat menunjukkan performa yang lebih konsisten.

DFS (Depth-First Search): Sebaliknya, DFS secara rata-rata lebih hemat dalam jumlah simpul yang dieksplorasi, namun membutuhkan waktu yang lebih lama per giliran. Sebaran waktu pencariannya juga lebih lebar, menunjukkan variabilitas performa yang lebih besar. Hal ini kemungkinan disebabkan oleh sifat DFS yang menelusuri jalur secara mendalam, yang dapat membuatnya "terjebak" dalam perhitungan yang panjang pada cabang pohon tertentu yang kompleks.

V. KESIMPULAN

Penelitian ini berhasil mengimplementasikan dan membandingkan dua strategi algoritma pencarian pohon, yaitu Depth-First Search (DFS) dengan pemangkasan heuristik dan Best-First Search (BB-BFS) berbasis antrian prioritas, sebagai solusi untuk mengatasi inefisiensi pendekatan brute force dalam pemilihan langkah catur. Berdasarkan hasil pengujian melalui simulasi permainan melawan bot catur, beberapa kesimpulan dapat ditarik.

Kedua algoritma menunjukkan karakteristik kinerja yang sangat berbeda, yang menyoroti adanya pertukaran (trade-off) fundamental antara kecepatan dan efisiensi komputasi. Algoritma BB-BFS terbukti lebih unggul dalam hal kecepatan rata-rata per giliran. Namun, kecepatan ini dicapai dengan menjelajahi jumlah simpul yang jauh lebih besar, menjadikannya lebih boros secara komputasi. Sebaliknya, algoritma DFS dengan pemangkasan secara signifikan lebih efisien karena hanya menjelajahi sebagian kecil dari jumlah simpul yang diperiksa oleh BB-BFS. Kehematan ini dibayar dengan variabilitas waktu yang lebih tinggi dan potensi penundaan yang ekstrem pada cabang permainan yang kompleks.

Dari hasil permainan, terlihat bahwa kedua algoritma masih mengalami kesulitan dan posisi papannya cenderung memburuk seiring waktu saat melawan lawan dengan ELO 1600. Hal ini menunjukkan bahwa meskipun strategi pencarian telah dioptimalkan, kualitas keputusan akhir sebuah mesin catur juga sangat bergantung pada kedalaman pencarian dan keakuratan fungsi evaluasi yang digunakan.

Kontribusi utama dari makalah ini adalah menyajikan perbandingan empiris dari dua paradigma pencarian yang berbeda untuk domain catur, yang didukung oleh hasil eksperimen dari program yang dibuat. Analisis ini memberikan wawasan praktis mengenai kekuatan dan kelemahan dari masing-masing pendekatan dalam aplikasi nyata. Untuk pengembangan di masa depan, penelitian dapat dilanjutkan dengan mengimplementasikan teknik pemangkasan yang lebih canggih seperti alpha-beta pruning, melakukan eksperimen dengan kedalaman pencarian yang lebih tinggi, serta menyempurnakan fungsi evaluasi untuk meningkatkan kualitas permainan secara keseluruhan.

LAMPIRAN

https://youtu.be/qBnDuTdfezY

lynaten/chess2 at dev

ACKNOWLEDGMENT

Terima kasih kepada pak Monte, pak Rinaldi sudah menjadi dosen yang membimbing saya selama mengerjaan makalah ini, baik dalam perkuliahan maupun di luar.

REFERENCES

[1] J. R. C. Program, "Chess Programming Wiki," [Online]. Available: https://www.chessprogramming.org/. [Diakses 24 Juni 2025].

[2] C. E. Shannon, "Programming a Computer for Playing Chess," Philosophical Magazine, vol. 41, no. 314, hlm. 256-275, 1950.

[3] H.xim, "Stockfish Evaluation Guide," [Online]. Available: https://hxim.github.io/Stockfish-Evaluation-Guide/. [Diakses 24 Juni 2025].

[4] Lynaten, "Chess," GitHub Repository, 2024. [Online]. Available: https://github.com/lynaten/chess. [Diakses 24 Juni 2025].

[5] Chess.com, "Analisis Permainan," [Online]. Available: https://www.chess.com/analysis. [Diakses 24 Juni 2025].

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, 24 Juni 2025

Nathanael Rachmat - 13523142