Application of Heuristic Move Ordering for Efficient Mate-in-N Resolution in Chess

Mikhael Benrael Tampubolon - 13524009
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: mikhaelbenrael@gmail.com , 13524009@std.stei.itb.ac.id

Abstract*— Penyelesaian masalah *Mate-in-N* dalam catur merupakan tantangan komputasional klasik dalam domain pencarian graf adversarial. Berbeda dengan pencarian jalur tunggal, penyelesaian catur menuntut evaluasi pohon keputusan berbasis *AND/OR graph*, di mana setiap langkah penyerang (*OR-node*) harus divalidasi terhadap seluruh kemungkinan respons balasan pihak bertahan (*AND-node*). Makalah ini mengevaluasi inefisiensi eksplorasi ruang keadaan menggunakan algoritma pencarian buta seperti *Breadth-First Search* (BFS) murni, dan mengusulkan optimalisasi pemangkasan simpul melalui *Heuristic Move Ordering* berbasis evaluasi mobilitas Raja lawan. Heuristik ini terinspirasi dari prinsip Warnsdorff (1823) yang memprioritaskan penelusuran pada langkah dengan opsi balasan paling sedikit. Menggunakan metrik *nodes visited* yang independen dari arsitektur perangkat keras, eksperimen komparatif pada sejumlah *puzzle Mate-in-2* dan *Mate-in-3* menunjukkan bahwa performa heuristik ini sangat bergantung pada topologi ruang keadaan catur tersebut. Pada skenario optimal (*best-case*), pengurutan cabang *OR-node* berhasil memangkas evaluasi ruang status secara ekstrem hingga 99,55% dibandingkan BFS murni. Namun, pada skenario terburuk (*worst-case*), heuristik mengalami degradasi menjadi *exhaustive search* tanpa adanya reduksi simpul (0%) dan justru memunculkan *overhead* komputasi. Hasil ini membuktikan bahwa intuisi manusia dalam mencari *forcing moves* dapat diterjemahkan menjadi fungsi heuristik untuk memandu *Iterative Deepening Depth-First Search* (IDDFS), meskipun efektivitas pemangkasan masifnya bersifat probabilistik dan tidak selalu terjamin pada setiap posisi graf.

Keywords—Mate-in-N, Heuristic Move Ordering, AND/OR Graph; Breadth-First Search.

I. PENDAHULUAN

Catur adalah permainan papan strategi dua pemain yang telah dimainkan selama berabad-abad dan dikenal sebagai salah satu permainan paling kompleks yang pernah ada. Kompleksitasnya menjadikan catur arena pengujian utama bagi algoritma pencarian graf dalam ilmu computer.

Gambar 1. Papan Catur diambil dari Chess.com

Dengan branching factor rata-rata 35 langkah per giliran, pohon pencarian tumbuh secara eksponensial sebesar $O(b^d)$, sehingga BFS murni pada kedalaman 5 ply harus mengeksplorasi hingga $35^5 \approx 52$ juta node, sehingga menjadikan blind search tidak praktis seiring bertambahnya kedalaman.

Sub-masalah Mate-in-N menuntut kebenaran mutlak, bukan estimasi probabilistik. Penyelesaiannya dimodelkan sebagai AND/OR graph: penyerang mencari satu rangkaian langkah yang benar (OR-node), sementara solusi tersebut harus tahan terhadap semua balasan legal pihak bertahan (AND-node).

Untuk mengatasi ledakan ruang pencarian tersebut, berbagai strategi pemangkasan telah dikembangkan, mulai dari Alpha-Beta Pruning hingga pendekatan berbasis heuristik domain-specific. Dalam konteks Mate-in-N, heuristik yang paling relevan adalah yang mampu mengidentifikasi langkah forcing yakni langkah yang secara agresif membatasi pilihan lawan sehingga cabang yang paling menjanjikan dieksplorasi lebih awal dan cabang yang tidak relevan dapat diabaikan lebih cepat.

Makalah ini mengusulkan Heuristic Move Ordering berbasis mobilitas Raja lawan untuk mengurutkan cabang OR-node sebelum dieksekusi IDDFS. Langkah yang paling mengekang ruang gerak lawan dievaluasi lebih dahulu. Kontribusi utamanya adalah pembuktian empiris bahwa teknik ini mampu memangkas nodes visited secara signifikan dibandingkan BFS, tanpa mengorbankan kebenaran logis solusi.

.

II. LANDASAN TEORI

A. Pemodelan Catur sebagai State-Space Graph

Dalam ilmu komputer, permainan catur direpresentasikan sebagai graf berarah:

$$G = (S, E, s_0, S_T)$$

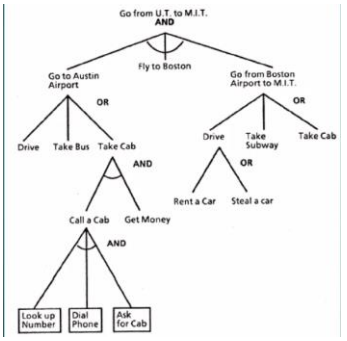
dengan S adalah himpunan semua state papan (posisi legal), E adalah himpunan sisi yang merepresentasikan langkah legal, s_0 adalah posisi awal, dan S_T adalah himpunan terminal state (skakmat atau seri). Shannon (1950) mengestimasi $|S| \approx 10^{43}$ posisi legal, sebuah angka yang jauh melampaui kapasitas memori maupun kecepatan komputasi mesin mana pun.

Berbeda dengan penelusuran graf standar seperti penemuan rute terpendek, graf catur bersifat turn-based dan dikendalikan oleh dua agen dengan tujuan bertolak belakang (zero-sum game). Dengan branching factor rata-rata $b \approx 35$, pertumbuhan eksponensial pohon pencarian menjadi hambatan utama yang harus diatasi oleh setiap solver.

B. Struktur AND/OR Graph dalam Adversarial Search

Karakteristik fundamental dari Mate-in-N solver adalah pendekatannya yang tidak dapat menggunakan algoritma single-agent (seperti A* atau Dijkstra) secara langsung. Masalah ini harus diselesaikan dalam kerangka AND/OR graph, sebagaimana dijelaskan oleh Russell dan Norvig:

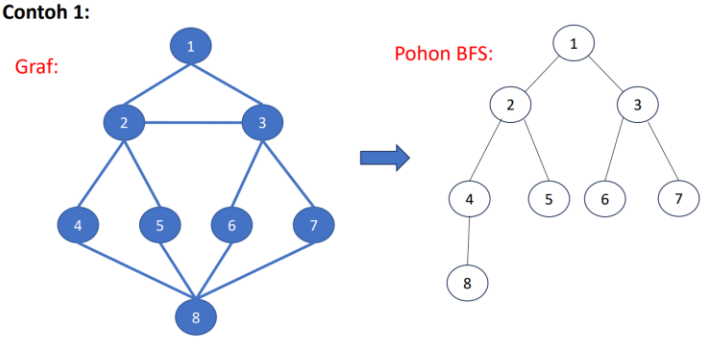
- **OR-Node (Giliran Penyerang/Attacker):** Pada posisi ini, algoritma bertindak sebagai Existential Quantifier (∃). Untuk membuktikan bahwa suatu simpul mengarah ke status "Menang", algoritma hanya perlu menemukan minimal satu langkah sah yang menghasilkan status menang. Jika satu saja cabang di bawahnya bernilai True, pencarian pada simpul siblings-nya dapat langsung dihentikan (cut-off), memungkinkan pruning masif.
- **AND-Node (Giliran Bertahan/Defender):** Pada posisi ini, algoritma bertindak sebagai Universal Quantifier (∀). Lawan diasumsikan bermain optimal untuk meloloskan diri dari ancaman skakmat. Sebuah simpul penyerang hanya terbukti valid jika seluruh balasan legal dari pihak bertahan tetap berujung pada status kalah (mat). Jika terdapat satu saja langkah balasan yang meloloskan diri, seluruh cabang penyerang tersebut gugur (False).



Gambar 2. AND/OR graph diambil dari [7]

C. Algoritma Breadth-First Search (BFS) dan Inefisiensinya

Algoritma *Breadth-First Search* (BFS) adalah metode penelusuran graf yang dilakukan secara transversal tingkat demi Tingkat. Algoritma ini memulai penelusuran dari simpul akar (*root*) dan mengunjungi seluruh simpul tetangga pada kedalaman *d* terlebih dahulu sebelum bergerak ke simpul pada kedalaman *d+1*.



Gambar 3. BFS graph diambil dari [2]

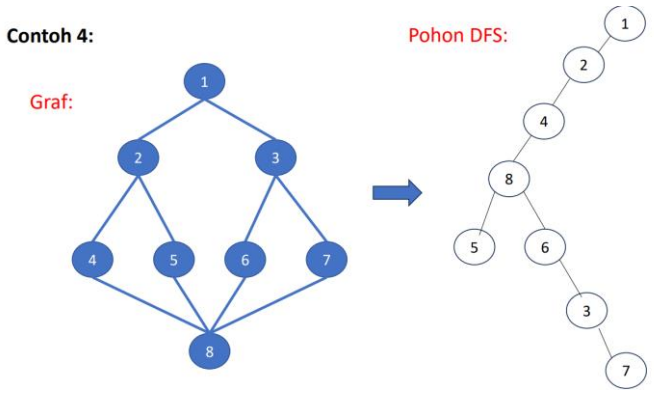
Untuk melacak simpul yang telah dibangkitkan, BFS memanfaatkan struktur data antrean (*FIFO Queue*). Secara umum, BFS menjamin penemuan jalur terpendek (*optimality*) pada graf tanpa bobot dan memiliki sifat *completeness* (pasti menemukan solusi jika ada). Namun, karakteristik ini harus dibayar dengan kompleksitas waktu $O(b^d)$ dan kompleksitas ruang $O(b^d)$, di mana b adalah *branching factor* dan d adalah kedalaman maksimum graf.

Dalam konteks penyelesaian masalah *Mate-in-N*, karakteristik penelusuran melebar ini menimbulkan inefisiensi yang masif. BFS akan membangkitkan semua kemungkinan langkah penyerang pada kedalaman 1, memverifikasinya terhadap semua balasan pada kedalaman 2, lalu mengeksplorasi langkah penyerang lagi pada kedalaman 3, dan seterusnya. Karena tidak ada panduan mengenai mana langkah yang paling mengancam (*forcing moves*), BFS dipaksa mengunjungi puluhan ribu hingga jutaan node yang sama sekali tidak relevan.

D. Algoritma Depth-First Search (DFS) dan Iterative Deepening (IDDFS)

. Berbeda dengan BFS, *Depth-First Search* (DFS) adalah algoritma penelusuran graf yang menjelajahi cabang graf

sejauh mungkin ke dalam sebelum melakukan proses runut-balik. DFS memanfaatkan struktur data tumpukan (*LIFO Stack*) atau dilakukan secara rekursif melalui *activation stack* pada memori komputer.



Gambar 3. DFS graph diambil dari [2]

Keunggulan utama DFS terletak pada efisiensi ruangnya, di mana kompleksitas ruangnya bersifat linier terhadap kedalaman, yaitu O(d). Walaupun begitu, pada graf yang sangat besar atau tak berhingga, DFS murni memiliki kelemahan fatal: algoritma ini tidak menjamin *completeness* karena rentan terjebak pada jalur yang salah secara mendalam, serta tidak menjamin solusi pada kedalaman minimum ditemukan terlebih dahulu.

Untuk menggabungkan keunggulan memori linier dari DFS dan jaminan optimalitas kedalaman dari BFS, Korf (1985) memformalkan algoritma *Iterative Deepening Depth-First Search* (IDDFS) . IDDFS bekerja dengan menjalankan *depth-limited DFS* secara berulang, dimulai dari batas kedalaman = 1, kemudian meningkat secara bertahap hingga solusi ditemukan. Dengan strategi ini, IDDFS menjamin solver selalu menemukan solusi pada kedalaman minimum tanpa mengonsumsi memori eksponensial layaknya BFS. Untuk masalah *Mate-in-N*, IDDFS dioperasikan dengan *depth budget* yang ketat dan tetap, sehingga *overhead* komputasi akibat iterasi awal menjadi sangat kecil dibandingkan manfaat penghematan ruang memori yang ditawarkannya.

E. Heuristik Mobilitas dan Prinsip Warnsdorff

Untuk mengatasi ledakan eksponensial, makalah ini memformulasikan implementasi Heuristic-Guided IDDFS. Alih-alih mengganti pencarian penuh dengan fungsi evaluasi (yang berisiko memberikan hasil salah), heuristik hanya digunakan pada ranah Move Ordering (Pengurutan Langkah).

Pendekatan ini terinspirasi dari Warnsdorff's Rule (1823) [5], sebuah heuristik historis untuk menyelesaikan Knight's Tour yang menyatakan: selalu pilih langkah menuju petak yang memiliki jumlah kemungkinan langkah lanjutan paling sedikit (minimal degree). Prinsip ini terbukti sangat efektif karena langkah dengan degree rendah lebih sulit dijangkau kemudian, sehingga sebaiknya dikunjungi lebih awal.

Dalam konteks Mate-in-N, prinsip serupa diterapkan pada OR-node penyerang: Mobilitas (M) didefinisikan sebagai jumlah langkah legal yang tersedia bagi Raja lawan setelah suatu langkah penyerang disimulasikan. Fungsi Heuristik H(s) dihitung sebagai:

$$H(s)=M(Raja\_Lawan(s'))$$

Semakin kecil nilai H(s), semakin terbatas opsi lawan, yang berarti langkah s tersebut sangat memaksa (forcing). Langkah-langkah pada OR-node diurutkan secara ascending berdasarkan skor H(s) ini sebelum pemanggilan rekursi AND-node dilakukan. Karena heuristik ini hanya mengatur urutan eksplorasi tanpa memotong pencarian secara permanen, kebenaran solusi akhir tetap terjamin.

III. METODOLOGI PENELITIAN

A. Batasan Eksperimen dan Isolasi Variabel

Untuk membuktikan kemanjuran algoritma tanpa terhalang kompleksitas rekayasa perangkat lunak dasar, eksperimen menggunakan pustaka python-chess yang menangani *move generator*, deteksi legalitas langkah, skak, dan skakmat secara otomatis. Pengujian dibatasi pada masalah Mate-in-2 (kedalaman maksimum 3 *ply* aktif) dan Mate-in-3 (kedalaman maksimum 5 *ply* aktif) untuk memastikan eksperimen selesai dalam waktu polinomial, serta menjamin sistem tidak kehabisan memori.

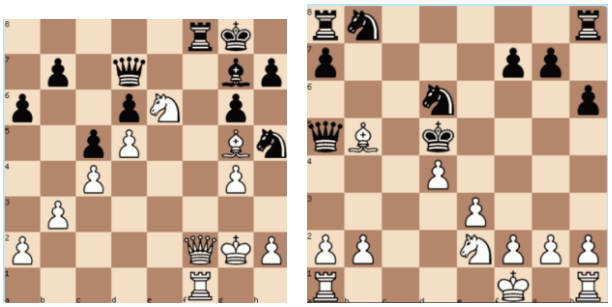
B. Deskripsi Puzzle Benchmark

Empat posisi *benchmark* dipilih dari potongan posisi *puzzle* (*tactical compositions*) dari partai resmi bertaraf internasional yang terdokumentasi dan dapat diverifikasi secara historis untuk memastikan reproduktibilitas eksperimen. Tabel II di bawah ini mendeskripsikan setiap puzzle:

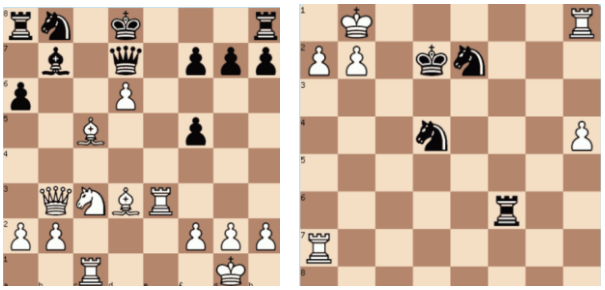
Tabel II. Deskripsi Posisi Puzzle Benchmark

Code	**Type**	**Position Description**	**Turn**
P1_Paulsen	Mate-in-2	Frankfurt 1878 (Wilfried Paulsen vs A. Anderssen)	White
P2_Dzagnidze	Mate-in-2	Tbilisi 2004 (Nana Dzagnidze vs I. Charkhalashvili)	White
P3_Shahade	Mate-in-3	Parsippany 2002 (Jennifer Shahade vs A. Feuerstein)	White
P4_Karpov	Mate-in-3	Baguio 1978 (Viktor Korchnoi vs Anatoly Karpov)	Black

P1 – P2:



P3 – P4:



C. Metrik Pengukuran (Nodes Visited)

Penilaian efektivitas algoritma secara sengaja tidak didasarkan pada hitungan *wall-clock time* (milidetik eksekusi mentah) secara eksklusif, karena hal ini bergantung pada *overhead* interpreter Python dan arsitektur perangkat keras. Metrik utama yang diamati adalah *Nodes Visited* (Jumlah Simpul Terkunjungi). Sebuah variabel global nodes_visited diinkrementasi tepat di baris teratas fungsi *solver*. Rasio perbedaan jumlah simpul ini merupakan representasi murni dari kemampuan pemangkasan struktur graf.

D. Implementasi Kode Pengurutan Cabang

Untuk menciptakan perbandingan yang adil terhadap algoritma heuristik, fungsi dasar BFS dimodifikasi menjadi penelusuran buta sempurna.

1. Algoritma Heuristic

```
def solve_mate_heuristic(board, depth, is_attacker):
    global nodes_visited
    nodes_visited += 1
    if board.is_checkmate():
        return True
    if depth == 0 or board.is_stalemate() or
board.is_insufficient_material():
        return False
    if is_attacker:
        legal_moves = list(board.legal_moves)
        move_scores = []
        for move in legal_moves:
            board.push(move)
            score = len(list(board.legal_moves))
            move_scores.append((score, move))
            board.pop()
        move_scores.sort(key=lambda x: x[0])
        for score, move in move_scores:
            board.push(move)
            result = solve_mate_heuristic(board, depth - 1,
False)
            board.pop()
            if result:
                return True
        return False
    else:
        for move in board.legal_moves:
            board.push(move)
            result = solve_mate_heuristic(board, depth - 1,
True)
            board.pop()
            if not result:
                return False
        return True
```

2. Algoritma BFS

```
def solve_mate_bfs(board, depth, is_attacker):
    global nodes_visited
    nodes_visited += 1
    if board.is_checkmate():
        return True
    if depth == 0 or board.is_stalemate() or
board.is_insufficient_material():
        return False
    if is_attacker:
        blind_moves = list(board.legal_moves)[::-1]
        for move in blind_moves:
            board.push(move)
            result = solve_mate_bfs(board, depth - 1, False)
            board.pop()
            if result:
                return True
        return False
    else:
        for move in board.legal_moves:
            board.push(move)
            result = solve_mate_bfs(board, depth - 1, True)
            board.pop()

            if not result:
                return False
return True
```

3. Main

```
if __name__ == "__main__":
    puzzles = {
        "P1_Paulsen                    (Mate-in-2)":
("1r4k1/1p1q2bp/p2pN1p1/2pP1bBn/2P3P1/1P6/P4QKP/5
R2 w - - 0 1", 3),
        "P2_Dzagnidze                  (Mate-in-2)":
("rn5r/p4pp1/3n3p/qB1k4/3P4/4P3/PP2NPPP/R4K1R w - -
0 1", 3),
        "P3_Shahade                    (Mate-in-3)":
("rn1k3r/1b1q1ppp/p2P4/2B2p2/8/1QNB1R2/PP3PPP/2R3
K1 w - - 0 1", 5),
"P4_Karpov                     (Mate-in-3)":
("2K4R/PP1kn3/8/4n2P/8/5r2/R7/8 b - - 0 1", 5)
    }
```

```
    print(f"{'PUZZLE':<25}  |  {'ALGORITMA':<12}  |  {'NODES VISITED':<15} | {'TIME (s)'}")
    print("-" * 75)

    for name, (fen, depth) in puzzles.items():

        is_white_turn = "w" in fen

        board = chess.Board(fen)
        nodes_visited = 0
        start_time = time.time()
        solve_mate_bfs(board, depth, is_attacker=True)

        time_bfs = time.time() - start_time
        nodes_bfs = nodes_visited

        board = chess.Board(fen)
        nodes_visited = 0
        start_time = time.time()

        solve_mate_heuristic(board, depth, is_attacker=True)
        time_heur = time.time() - start_time
        nodes_heur = nodes_visited

        pruning_percent = ((nodes_bfs - nodes_heur) / nodes_bfs) * 100 if nodes_bfs > 0 else 0
        print(f"{name:<25}  |  {'BFS Pure':<12}  |  {nodes_bfs:<15,}  |  {time_bfs:.4f}")
        print(f"{'':<25}  |  {'Heuristic':<12}  |  {nodes_heur:<15,}  |  {time_heur:.4f}")
        print(f"{'--> Pruning Effect':<25}  |  {'':<12}  |  {pruning_percent:.2f}% reduction\n")
```

Urutan langkah sah bawaan *engine* dibalik secara sengaja ([::-1]) agar BFS mengeksplorasi graf dalam skenario terburuk (*worst-case scenario*). Di sisi lain, algoritma Heuristik mengevaluasi dan mengurutkan ulang setiap simpul OR-node secara *ascending* berdasarkan jumlah mobilitas legal lawan sebelum dieksekusi rekursif.

IV. HASIL & PEMBAHASAN

A. Hasil Eksperimen Komparasi Algoritma

Eksperimen komputasi dieksekusi pada keempat posisi benchmark. Tabel I memaparkan data empiris perbandingan nodes visited serta persentase reduksi ruang pencarian.

Tabel I. Perbandingan Nodes Visited: BFS vs. Heuristic Ordered IDDFS

Puzzle	Type	BFS Pure (Nodes)	Heuristic (Nodes)	Reduksi
P1	Mate-in-2	2,376	2,376	0.00% ↓
P2	Mate-in-2	889	4	99.55 % ↓
P3	Mate-in-3	145,714	145,714	0.00% ↓
P4	Mate-in-3	3,939	3,914	0.63% ↓

B. Analisis Kasus Pemangkasan Ekstrem (Best-Case Scenario)

Pada *puzzle* P2_Dzagnidze, hipotesis teoritis mengenai pemangkasan ruang graf terbukti secara mutlak. BFS murni terpaksa membangkitkan 889 simpul, sedangkan heuristik mobilitas berhasil menemukan skakmat hanya dengan mengeksplorasi 4 simpul (reduksi 99,55%). Fenomena ini terjadi akibat *cut-off* dini pada representasi logika OR-node. Langkah yang paling membatasi gerak Raja lawan terurut pada indeks pertama. Saat langkah tersebut mengembalikan nilai *True* (skakmat tercapai), evaluasi terhadap ratusan *siblings* (cabang persaudaraan) langsung dihentikan dan dipangkas dari memori.

C. Analisis Exhaustive Search dan Batas Heuristik (Worst-Case Scenario)

Hasil eksperimen pada P1_Paulsen dan P3_Shahade menunjukkan persentase reduksi sebesar 0.00%, di mana Heuristik mengunjungi jumlah simpul yang sama persis dengan BFS murni (hingga 145.714 simpul pada P3). Secara analitis, hal ini merepresentasikan kondisi *worst-case* dalam arsitektur AND/OR graph. Meskipun heuristik telah mengurutkan langkah penyerang secara cerdas, struktur graf pada posisi khusus tersebut memaksa algoritma untuk memverifikasi seluruh kemungkinan balasan lawan (AND-node) untuk membuktikan tidak ada jalan lolos. Selain itu, jika jalur solusi sejati tidak memicu skor mobilitas terendah pada *ply* pertama, heuristik akan mengeksplorasi cabang-cabang lain terlebih dahulu hingga menemui jalan buntu (mengembalikan *False*). Karena algoritma dijamin *complete*, ia pada akhirnya tetap mengunjungi cabang solusi secara utuh (exhaustive search), menghasilkan ukuran pohon yang tidak mengalami reduksi.

D. Analisis Kompleksitas dan Overhead Komputasi

Eksperimen membuktikan bahwa dalam skenario *best-case* (P2), performa penelusuran mampu melampaui batas efisiensi teoritis *alpha-beta pruning* $O(b^{d/2})$. Namun, terdapat *trade-off* operasional yang signifikan. Untuk melakukan kalkulasi heuristik, mesin harus melakukan simulasi *push()* dan *pop()* pada setiap simpul semu hanya untuk menghitung panjang senarai *legal_moves* lawan. Pada graf berskala besar di mana *pruning* gagal terjadi (seperti P3_Shahade), kalkulasi primitif ini menciptakan *overhead* waktu komputasi yang substansial (dari 0.74 detik menjadi 4.66 detik), meskipun jumlah *nodes visited* konstan. Ini menunjukkan bahwa kecerdasan heuristik memiliki beban komputasional bawaan yang harus dikompensasi dengan *cut-off* yang sukses.

Heuristic Move Ordering juga menambah overhead komputasi dari proses pengurutan. Fungsi sort() di Python yang menggunakan Timsort memiliki kompleksitas

$O(k \log k)$, dengan k adalah jumlah langkah legal (sekitar 30–35 per posisi). Meskipun kecil secara teoritis, pemanggilan berulang pada banyak node (OR-node) menyebabkan akumulasi waktu yang signifikan. Hal ini menjelaskan mengapa pada puzzle P3_Shahade, waktu eksekusi heuristik mencapai 4,66 detik, lebih lambat dibanding BFS murni sebesar 0,74 detik, meskipun jumlah node yang dieksplorasi sama.

V. KESIMPULAN

Penyelesaian Mate-in-N merepresentasikan persoalan *adversarial graph traversal* berbasis struktur AND/OR *node*. Eksperimen menggunakan algoritma penelusuran buta (seperti BFS dengan *reversed move generation* memvalidasi bahwa ledakan eksponensial $O(b^d)$ lazim terjadi, bahkan pada kedalaman minimal seperti 3 *ply* dan 5 *ply*.

Implementasi *Heuristic Move Ordering* berbasis evaluasi mobilitas Raja lawan terbukti bukan merupakan peluru perak (*silver bullet*), melainkan strategi probabilistik yang efisiensinya sangat bergantung pada topologi ruang keadaan catur tersebut. Pada skenario optimal (*best-case*), pengurutan prioritas langkah-langkah *forcing* mampu mengeksekusi *pruning* secara instan dan memangkas pohon keputusan hingga 99,55%. Namun, pada skenario terburuk (*worst-case*) di mana jalur solusi sejati bukan merupakan manuver dengan pembatasan mobilitas tertinggi di langkah awal, fungsionalitas algoritma terdegradasi menjadi *exhaustive search*, menghasilkan ukuran graf yang identik dengan pencarian buta (reduksi 0.00%) ditambah dengan beban *overhead* waktu kalkulasi heuristik.

Penelitian selanjutnya disarankan tidak menggunakan heuristik mobilitas sebagai metode tunggal, melainkan mengintegrasikannya dengan Alpha-Beta Pruning untuk memperoleh cut-off yang lebih optimal. Selain itu, efisiensi pada kasus worst-case dapat ditingkatkan dengan penggunaan Transposition Table, yang bekerja seperti memoization pada Dynamic Programming. Dengan menyimpan hasil evaluasi setiap board state menggunakan hash (misalnya Zobrist Hashing), algoritma dapat menghindari evaluasi ulang pada posisi yang sama akibat transposisi. Kombinasi Heuristic Move Ordering dan Transposition Table diharapkan dapat mengurangi overhead pencarian secara signifikan dibandingkan blind search.

ACKNOWLEDGMENTS

Penulis mengucapkan puji syukur kepada Tuhan Yang Maha Esa atas rahmat dan perlindungan-Nya sehingga makalah ini dapat diselesaikan dengan baik. Penulis juga menyampaikan terima kasih kepada Dr. Ir. Rinaldi, M.T., selaku dosen pengampu mata kuliah IF2211 Strategi Algoritma, atas bimbingan, arahan, dan dukungan yang telah diberikan selama perkuliahan berlangsung.

REFERENCES

[1] C. E. Shannon, "Programming a Computer for Playing Chess," *Philosophical Magazine*, Ser. 7, vol. 41, no. 314, pp. 256–275, 1950.

[2] R. Munir, "BFS-DFS (2026) Bagian 1," Mata Kuliah IF1220 Matematika Diskrit, 2025–2026. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/13-BFS-DFS-(2026)-Bagian1.pdf, diakses 19 Juni 2026 pukul 23.00.

[3] R. Munir, "BFS-DFS (2026) Bagian 2," Mata Kuliah IF1220 Matematika Diskrit, 2025–2026. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/14-BFS-DFS-(2026)-Bagian2.pdf, diakses 19 Juni 2026 pukul 23.00.

[4] R. Munir, "Route Planning (2026) Bagian 1," Mata Kuliah IF1220 Matematika Diskrit, 2025–2026. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/21-Route-Planning-(2026)-Bagian1.pdf, diakses 19 Juni 2026 pukul 23.00.

[5] R. Munir, "Route Planning (2026) Bagian 2," Mata Kuliah IF1220 Matematika Diskrit, 2025–2026. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/22-Route-Planning-(2026)-Bagian2.pdf, diakses 19 Juni 2026 pukul 23.00.

[6] N. Fiekas, *python-chess: A Chess Library for Python*. [Daring]. Tersedia: https://python-chess.readthedocs.io/, diakses 10 Juni 2026.

[7] A. A. Jalil, "AND-OR Graph," Slideshare. [Daring]. Tersedia: https://www.slideshare.net/slideshow/and-or-graph/236595990, diakses 19 Juni 2026 pukul 23.00.

[8] Chess.com, "Mate in Three Puzzles." SparkChess. [Daring]. Tersedia: https://www.sparkchess.com/mate-in/three, diakses 19 Juni 2026 pukul 23.00.

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, 19 Juni 2026

Mikhael Benrael Tampubolon 13524009