https://journal.ubm.ac.id/index.php/alu
Jurnal ALU

Jurnal Algoritma, Logika dan Komputasi
Vol.I (No. 2) : 68 - 81. Th. 2018
p-ISSN: 2620-620X
e-ISSN: 2621-9840

PERBANDINGAN METODE OPTIMASI ALGORITMA *MINIMAX* PADA PERMAINAN CATUR

Comparison Of Minimax Algorithm Optimization Methods In Chess Games

Alfarabi Dwi Karuniawan, alfarabidwik@gmail.com [1)], Aries Saifudin, aries.saifudin@gmail.com [2)] dan Ari Irawan, ari_irawan@tau.ac.id [3)]
[1)] *Program Studi Teknik Informatika, Universitas Pamulang*
[2)] *Program Studi Teknik Informatika, Universitas Pamulang*
[3)] *Program Studi Sistem Informasi, Universitas Tanri Abeng*

ABSTRACT

Chess games have been widely developed on digital media, but the games that have been produced still have some shortcomings, such as the slow computer to determine the next movement, or less precisely the computer chooses the piece and its movement when match versus human. In previous research on artificial intelligence, chess games can be created using the minimax algorithm, but the complexity of chess game causing a slow process when determining the movement of pieces. In this research porposed some methods such as alpha-beta pruning, capturing heuristic, transposition tables, and history heuristics to optimize the best movement search speed on a chess game that utilizes the minimax algorithm as a basic search method. Based on the implementation and testing, minimax algorithm can optimize the search speed to find out the best moves.

Keywords : *chess, game, search, minimax, heuristic*

ABSTRAK

Permainan catur telah banyak dikembangkan pada media digital, namun permainan yang telah dihasilkan masih memiliki beberapa kekurangan, seperti lambatnya komputer menentukan pergerakan lanjutan, atau kurang tepatnya komputer memilih bidak dan pergerakannya saat bertanding melawan pengguna. Pada penelitian terdahulu tentang kecerdasan buatan, permainan catur dapat dibuat dengan menggunakan algoritma minimax, namun karena kompleksitas dari permainan catur yang mengakibatkan lambatnya proses saat komputer menghitung langkah dan mengambil keputusan pergerakan bagi bidak-bidak miliknya, beberapa metode seperti alpha-beta pruning, capturing heuristic, transposition tables, dan history heuristic telah dibuat untuk mengoptimasi kecepatan pencarian gerakan terbaik pada permainan catur yang memanfaatkan algoritma minimax sebagai dasar metode pencarian. Berdasarkan penerapan dan pengujian dalam penelitan ini, metode-metode tersebut dapat mengoptimasi kecepatan pencarian untuk mengetahui pergerakan terbaik yang sebelumnya hanya memanfaaatkan algoritma minimax.

Kata kunci: *Catur, Permainan, pencarian, minimax, heuristis*

PENDAHULUAN

Permainan catur merupakan permainan yang sangat populer pada abad terakhir [1, p. 29]. Permainan catur sering dimainkan oleh berbagai kalangan masyarakat sebagai permainan yang menyenangkan, menantang, dan dapat membantu mengasah otak para pemainnya. Permainan catur dapat membuat anak-anak lebih cerdas, dapat meningkatkan kemampuan matematika, dan dapat meningkatkan kemampuan akademik, serta banyak klaim serupa yang telah dibuat tentang manfaat catur dalam mengembangkan pendidikan [2, p. 124]. Permainan catur dapat meningkatkan kemampuan di sekolah dalam bidang matematika dan bahasa Rumania [3]. Pengajaran catur dapat

Jurnal Algoritma, Logika dan Komputasi
Vol.I (No. 2) : 68 - 81. Th. 2018
p-ISSN: 2620-620X
e-ISSN: 2621-9840

meningkatkan secara signifikan kemampuan matematika dan kapasitas metakognitif siswa usia sekolah [4, p. 1]. Selain itu, permainan catur telah resmi menjadi cabang olahraga, sehingga secara periodik seringkali dibuatkan ajang kompetisi catur baik berskala nasional maupun internasional.

Permainan catur adalah permainan dengan struktur aturan yang kompleks, dan tingkat individual permainannya tergantung pada representasi sistem peraturannya, serta antisipasi dan kreatifitas yang digunakan selama permainan [5, p. 138]. Permasalahan pada permainan catur adalah kompleksitas strategi yang dapat dibuat dan banyaknya jumlah pergerakan yang mungkin dapat dilakukan oleh kedua pemain selama permainan berlangsung. Sebagai ilustrasi, ketika salah satu pemain akan menggerakkan salah satu bidak, maka pemain tersebut akan menduga-duga apa yang akan dilakukan oleh pemain lawan setelahnya. Kemudian dia akan berpikir kembali apa yang akan dilakukan olehnya setelah lawannya melangkah, dan seterusnya hingga merasa cukup. Setiap mengambil keputusan untuk melangkah, pemain selalu mempertimbangkan keuntungan/kerugian yang akan didapat, karena dalam permainan catur, setiap bidak memiliki gerakan logika dan ukuran langkah yang berbeda [6, p. 4].

Kecerdasan buatan merupakan salah satu disiplin ilmu yang menjadi dasar pengetahuan untuk mengembangkan permainan ke dalam media digital. Di antara beberapa teknik dalam kecerdasan buatan yang dapat mewujudkan permainan pada media digital adalah teknik pohon pencarian [7, p. 13]. Fungsi evaluasi dan mekanisme pencarian merupakan komponen utama dari setiap program catur [8, p. 782]. Salah satu algoritma yang dibuat untuk melakukan prosedur pencarian pada kecerdasan buatan dan dapat diterapkan pada permainan satu lawan satu seperti catur adalah algoritma minimax. Algoritma minimax telah diciptakan sejak tahun 1950, yaitu saat Shannon melakukan penelitian untuk mencari solusi terhadap kompleksitas evaluasi pada permainan satu lawan satu [9, pp. 256-257].

Permasalahan dalam implementasi algoritma minimax ke dalam permainan catur timbul karena algoritma ini mampu mencari seluruh kemungkinan pergerakan yang dapat dilakukan bagi kedua pemain, sehingga jumlah pencarian yang diproses oleh komputer menjadi terlalu banyak sehingga durasi saat proses tersebut dieksekusi menjadi terlalu lambat. Dapat dilihat pada Tabel 1, bahwa total kemungkinan pergerakan yang disimbolkan dengan (Y) dapat menjadi begitu banyak saat kedalaman level pencarian (N) bertambah. Hal ini menyebabkan teknik pohon pencarian untuk mengetahui pergerakan terbaik yang dieksekusi pada suatu perangkat lunak permainan catur menjadi lambat.

Table 1. Prediksi pencarian

N	P	JB	X	Y $y_{(n-1)}*x_{(n)}$
1	W E2:E4	16	B 20	20
2	B E7:E5	16	W 20	400
3	W D1:F3	16	B 28	11.200
4	B D8:G5	16	W 28	313.600
5	W B1:C3	16	B 37	11.603.200
6	B B8:A6	16	W 32	371.302.400

N = kedalaman level, P = pergerakan, W = putih, B = hitam , JB = jumlah bidak, X = kemungkinan gerakan masing-masing bidak, Y = total kemungkinan dalam pohon pencarian.

Beberapa metode dapat digunakan untuk mengoptimasi algoritma minimax dengan memangkas banyaknya jumlah pencarian yang terdapat pada pohon pencarian. Metode-metode tersebut seperti algoritma *alpha-beta pruning*, *best first SSS (Search Space Sampling)*, *FSSS (Forward Search Space Sampling), MTD(f)* dan lain-lain [10, pp. 155-156].

Alpha-beta pruning sebagai salah satu metode perbaikan yang dapat mengoptimasi kecepatan pencarian pada algoritma *minimax*, algoritma ini merupakan suatu teknik yang melakukan pemangkasan pada simpul-simpul yang dianggap tidak penting pada pencarian algoritma *minimax*, sehingga jumlah pencarian yang dihasilkan

Jurnal Algoritma, Logika dan Komputasi
Vol.I (No. 2) : 68 - 81. Th. 2018
p-ISSN: 2620-620X
e-ISSN: 2621-9840

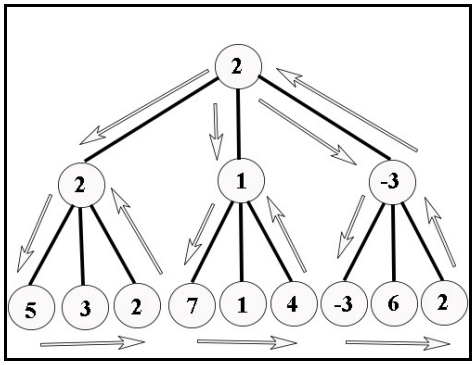
menjadi lebih sedikit dibanding dengan jumlah pencarian yang dihasilkan oleh algoritma *minimax* tanpa pemangkasan [11, p. 297]. Pada penelitian Liu, Zang dan Fu yang dilakukan terhadap pengembangan permainan catur dari kebudayaan cina yang mirip dengan permainan catur pada umumnya, metode *capturing heuristic, transposition table* dan *history heuristic* telah digunakan sebagai teknik tambahan terhadap *alpha-beta pruning* [12, pp. 5805-5807].

Pada penelitian kali ini kami mengusulkan algoritma *alpha-beta pruning*, *bubble sort* yang dimanfaatkan sebagai teknik pengurutan serupa dengan *capturing heuristic*, transposition table dan history heuristic yang memanfaatkan objek *hashmap* hasil dari algoritma *hashing* yang terdapat dalam *library* dari bahasa pemrograman java [13, p. 415].

Paper ini akan terbagi menjadi 5 bagian, bagian 2 menjelaskan teori-teori dari penelitian dengan metode serupa yang telah dilakukan oleh peneliti terdahulu, bagian 3 menjelaskan teori dari metode yang diusulkan dalam penelitian ini, bagian 4 menjabarkan hasil dan pembahasan dari penelitian, bagian 5 merupakan pernyataan-pernyataan kesimpulan yang dibuat dari hasil penelitian ini, dan bagian 6 adalah daftar pustaka yang dipelajari dalam penelitian ini.

Permasalahan permainan catur yang begitu kompleks menyebabkan banyaknya jumlah simpul dalam pohon pencarian yang menggunakan algoritma minimax sebagai teknik pencarian. Algoritma *minimax* adalah sebuah teknik yang tersusun dengan dasar dari teknik *Depth First Search* (DFS), yaitu suatu teknik berupa pohon pencarian dengan melakukan evaluasi mulai dari *node* paling kiri lalu menelusuri pada *subnode* di bawahnya hingga pencarian terdalam, dan lalu mengevaluasi setiap kemungkinan-kemungkinan lain yang ada di kanannya lalu kembali satu *node* ke atas dan mencari lagi kemungkinan lain sebelah kanan, kembali ke atas dan seterusnya hingga menemui semua kemungkinan dan mendapatkan nilai terbaik hasil dari perbandingan seluruh evaluasi pada pohon pencarian [7, pp. 18-19]. Prosedur yang dimulai dari awal pencarian hingga proses pencarian selesai yang diperlukan dalam penerapan algoritma *minimax* adalah [14, p. 435]:

1. Tandai setiap ke dalaman *node* pencarian dengan level, semakin dalam *node* pencarian maka semakin besar level dan jumlah *node* yang kemungkinan ditemukan dalam pencarian.
2. Buat fungsi evaluasi untuk permainan, dan evaluasi pada *node* paling bawah mulai dari sebelah kiri hingga kanan dan berikan nilai pada masing-masing *node* tersebut.
3. Setelah masing-masing *node* terbawah diberikan nilai, arahkan fungsi evaluasi pada *node* di atasnya, jika *node* di atasnya adalah langkah untuk max maka pilih nilai terbesar dari *node-node* yang telah dicari, berikan nilai tersebut kepada *node* di atasnya.
4. Jika *node* di atasnya adalah langkah min maka pilih nilai terkecil dari *node-node* yang telah dicari, berikan nilai tersebut kepada *node* di atasnya.
5. Lakukan hal yang sama dari no 3 atau 4 pada *node-node* di sebelah kanan dari *node* terakhir yang diberi nilai dan seterusnya, sehingga nilai terpilih dari *node* anak berpindah semakin ke atasnya, akhir *node* yang paling atas akan diberi nilai terpilih dari seluruh evaluasi pada pohon pencarian.



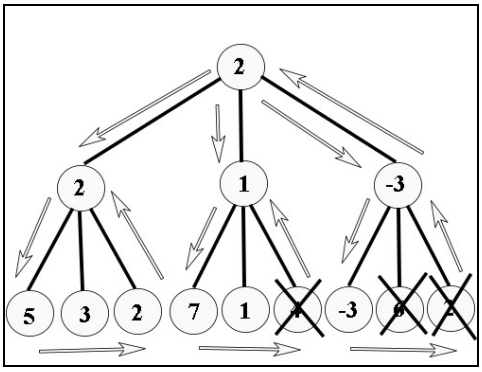
Gambar 1. Ilustrasi algoritma *minimax* dari kiri bawah ke kanan lalu ke atas

Jurnal Algoritma, Logika dan Komputasi
Vol.I (No. 2) : 68 - 81. Th. 2018
p-ISSN: 2620-620X
e-ISSN: 2621-9840

Pada penelitian Knuth dan More menjelaskan bahwa algoritma *minimax* dapat digunakan untuk mencari solusi terbaik di antara sebanyak apapun pilihan keputusan, mereka menyatakan tidak ada *infinite sequence* (perulangan tak terhingga) pada seluruh komposisi gerakan yang diperbolehkan bagi seluruh bidak, setiap perulangan pencarian terhadap posisi bagi bidak pasti akan mencapai jumlah tertentu, hanya saja macam komposisi pergerakan pada permainan catur dapat menjadi banyak jumlahnya untuk dievaluasi [11, p. 302]. Pada penelitian Liu, Zang dan Fu [12] mengusulkan metode *alpha-beta pruning*, *capturing heuristic*, *transposition table* dan *history heuristic* yang diimplementasikan untuk memperkecil jumlah simpul pencarian dan mengoptimasi lambatnya proses pencarian yang menggunakan algoritma *minimax* dalam permainan catur.

Sejak tahun 1958 para peneliti menganalisa permasalahan banyaknya jumlah pencarian dalam algoritma *minimax*, penelitian itu dilakukan oleh McCarthy dan kawan-kawannya di MIT, hasil dari penelitian-penelitian yang dilakukan saat itu menciptakan metode baru yaitu algoritma *alpha-beta prunning* [15, pp. 86-88]. *Alpha-beta prunning* tercetus berdasarkan pada pendekatan bahwa dalam pencarian terdapat *node-node* yang memiliki dua kemungkinan nilai, yaitu *optimistic value* dan *pessimistic value,* yang ditransisikan menjadi nilai *alpha* sebagai batasan maximum yang disimbolkan dengan α dan *beta* sebagai batasan minimum yang disimbolkan dengan β, batasan α dan β yang dibuat digunakan untuk membuktikan bahwa suatu perubahan terhadap keadaan tertentu tidak dapat mempengaruhi hasil pencarian [16, pp. 194-195].

Sebagai contoh diilustrasikan di mana F = fungsi mencari nilai terbesar, G = fungsi pencari nilai terkecil, P = node-node dalam pohon pencarian, jika pada prosedur algoritma *minimax* yang dilakukan menghasilkan nilai $F(p_1) = -10$ dan $G(p) >= 10$ dan tidak diketahui nilai G(p) dari $F(p_2)$, namun dapat diketahui nilai pertama pada pohon pencarian $F(p_2)$ dan nilai pergerakan yang diperbolehkan pada $F(p_{21}) <= 10$ maka fungsi evaluasi tidak perlu mencari semua *subnode* dari $F(p_2)$. Prosedur *branch and bound* ini menyelesaikan masalah *pessimistic value* dan *optimistic value* yang menghasilkan algoritma *alpha-beta pruning* yang memiliki 2 nilai *bound* yaitu *alpha* dan *beta*.



Gambar 2. Ilustrasi pemangkasan *alpha-beta pruning* terhadap *minimax*

Capturing heuristic didasari dari teknik pengurutan bertipe *descending* yang dapat mempercepat tercapainya nilai batas *alpha* dan *beta*, *sedangkan transposition table* dan *history heuristic* yang diimplementasikan dalam penelitian Liu, Zang dan Fu memanfaatkan algoritma *hashing* yang melakukan penyimpanan dan pembandingan data dari *node* yang sama yang terdapat dalam pohon pencarian. Ide mengenai *transposition table* timbul berdasarkan pemikiran bahwa *node-node* pada pohon pencarian yang terbentuk tidak selalu berbeda satu sama lain, seringkali antara satu *node* dengan *node* yang lain mewakili kondisi dan posisi yang sebenarnya sama [12, p. 5805]. Saat pencarian dilakukan di mana ditemukan *node* yang memiliki kesamaan posisi dengan yang telah tersimpan maka acuan nilai yang akan dibandingkan berdasarkan nilai pada *node* yang sama dalam tabel riwayat yang tersimpan, sedangkan *history heuristic* timbul berdasarkan pemikiran bahwa langkah-langkah lanjutan yang dipilih oleh pemain terkadang bukanlah langkah terbaik yang telah terprediksi oleh pohon pencarian [12, p. 5805], hanya dengan mengetahui data-data pencarian sebelumnya maka akan diketahui apakah

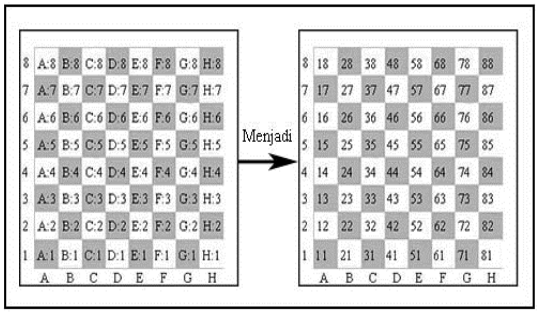
Jurnal Algoritma, Logika dan Komputasi
Vol.I (No. 2) : 68 - 81. Th. 2018
p-ISSN: 2620-620X
e-ISSN: 2621-9840

langkah pemain adalah langkah yang lebih buruk dari langkah terbaik yang telah terprediksi oleh pencarian sebelumnya, jika langkah tersebut adalah langkah yang lebih buruk maka perangkat lunak dapat melakukan gerakan terbaik yang telah diketahui sebelumnya, untuk mempercepat pengaksesan data yang telah tersimpan maka mekanisme penyimpanan dalam penelitian tersebut menggunakan *hashtable* atau *hashmap.*

METODE PENELITIAN

Dalam penelitian ini diusulkan metode *alpha-beta pruning*, *capturing heuristic* dengan menggunakan algoritma *bubble sort* sebagai teknik pengurutan, *transposition table* dan *history heuristic* yang diimplementasikan dengan tujuan sebagai perbandingan metode-metode optimasi terhadap algoritma *minimax*.

Agar setiap keadaan atau posisi dalam permainan dapat diberi nilai sebagai perbandingan evaluasi dalam pohon pencarian, maka objek-objek dalam permainan catur harus dapat diberi nilai-nilai tertentu yang mewakili aturan-aturan dalam permainan. Setiap petak dalam papan catur yang diberi penyebut kata seperti A:1, E:5 dan lain-lain akan ditransisikan kedalam fungsi evaluasi menjadi bilangan.

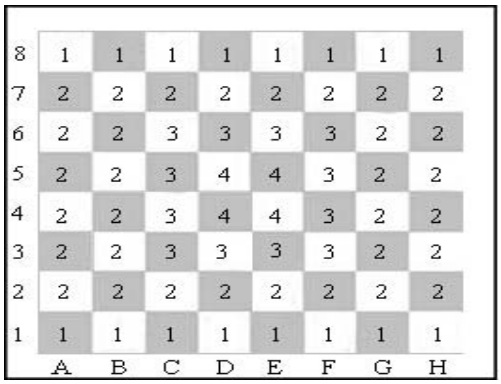


Gambar 3. Transisi penyebut kata menjadi penyebut bilangan terhadap petak catur

Jenis-jenis bidak dalam permainan catur dapat ditandai dengan memberikan nilai berbeda bagi masing-masing jenisnya, dalam penelitian ini jenis-jenis bidak akan dibedakan dengan memberikan nilai terhadap bidak raja = 6, ratu = 5, benteng = 4, pluncur = 3, kuda = 2, dan pion = 1, namun pemberian nilai tersebut tidak dapat digunakan sebagai pembeda antara bidak yang jenisnya sama, sedangkan dalam permainan catur pembukaan ada sebanyaknya 16 bidak berjenis sama, agar kunci identifikasi bidak menjadi unik, kunci-kunci tersebut akan diberi nilai komposisi dari nilai masing-masing bidak dan nilai posisi awal atau posisi pertama pada masing-masing kolom saat permainan baru dimulai, komposisi posisi awal dapat memberikan perbedaan nilai kunci antara satu bidak dengan bidak lain, untuk menghitung kunci bagi masing-masing bidak dapat dirumuskan dalam bentuk $K = B * 100 + NP$ Dimana *K* = kunci, *B* = nilai bidak dan *NP* = nilai posisi awal, sehingga jika *B = 2* dan *NP = 52* maka 2*100+52 = 252.

Tabel 2. Contoh pendefinisian kunci terhadap bidak

No	Jenis	Milik	Posisi	Hitung	Hasil Kunci
1	Pion	Putih	52	1*100+52	152
2	Pion	Putih	62	1*100+62	162
3	Pion	Hitam	57	1*100+57	157
4	Raja	Putih	51	6*100+51	651
5	Raja	Hitam	58	6*100+58	658



Gambar 4. Nilai posisi masing-masing kolom pada papan catur

Bidak yang menguasai wilayah tengah pada papan catur memiliki keunggulan lebih dibanding bidak yang berada pada posisi pinggir papan catur [17, p. 179]. Selain pernyataan tersebut, Raymond dan Stanley juga menjelaskan alasan pentingnya

Jurnal Algoritma, Logika dan Komputasi
Vol.I (No. 2) : 68 - 81. Th. 2018
p-ISSN: 2620-620X
e-ISSN: 2621-9840

penguasaan tengah pada papan catur, di mana ketika sebuah bidak menguasai lini tengah pada papan catur maka bidak tersebut memiliki banyak kemungkinan untuk melakukan serangan dibanding jika bidak tersebut berada di luar dari kolom tengah papan catur [18, pp. 99-103].

Dalam penelitian ini setiap pemain yang menempatkan bidaknya di atas petak-petak pada papan catur akan mendapat nilai tambahan sesuai dengan akumulasi dari seluruh nilai pada petak tempat masing-masing bidak berada, perhitungan nilai evaluasi posisi bagi masing-masing pemain yaitu N = (A*4) + (B*3) + (C*2) + (D*1) di mana:

A = {ajumlah bidak yang berada pada kolom bernilai 4}
B = {bjumlah bidak yang berada pada kolom bernilai 3}
C = {cjumlah bidak yang berada pada kolom bernilai 2}
D = {djumlah bidak yang berada pada kolom bernilai 1}

Saran-saran yang diberikan oleh para maestro permainan catur adalah dengan menempatkan bidak pada posisi yang memberikan kebebasan pergerakan, anjuran ini biasa dimanfaatkan oleh para pemain pada saat pembukaan permainan, di mana masing-masing pemain berusaha dengan segera menguasai lini tengah agar dapat lebih leluasa bergerak [17, p. 61], sehingga dapat disimpulkan semakin banyak jumlah gerakan yang mungkin dapat dilakukan maka semakin besar kekuatan yang dihasilkan oleh fleksibilitas pergerakannya, maka dengan menjumlahkan hasil evaluasi kolom (posisi bidak pada petak) dengan jumlah pergerakan bagi seluruh bidak di mana *EK* = nilai evaluasi petak, *JP* = jumlah pergerakan, *W* = pemain putih dan *B* = pemain hitam, hasil evaluasi EP = evaluasi posisi dapat dihasilkan dari rumus berikut:

$$EP_{(w)} = EK_{(w)} + JP_{(w)}$$
$$EP_{(b)} = EK_{(b)} + JP_{(b)}$$

Sebagai bentuk pemahaman terhadap pentingnya bidak-bidak dalam catur, umumnya para pemain catur tradisional juga memberikan tingkatan hirarki kepada masing-masing bidak, semakin tinggi hirarki bidak tersebut maka semakin penting keberadaannya bagi pemiliknya, bentuk hirarki bagi masing-masing bidak dijelaskan secara terurut dari mulai bidak yang paling penting hingga bidak yang tidak terlalu penting, yaitu [19, p. 8]:

a. Raja, merupakan bidak terpenting dalam permainan, jika raja dari salah satu pemain terbunuh maka permainan dinyatakan berakhir dan pemain yang masih memiliki raja dinyatakan menang.
b. Ratu lebih kuat daripada benteng, kuda dan pluncur.
c. Benteng lebih kuat daripada kuda dan pluncur.
d. Kuda dan pluncur memiliki kekuatan sama, namun untuk beberapa situasi pluncur dapat menjadi lebih unggul daripada kuda.
e. Bidak pion adalah bidak paling lemah.

Tabel 3. Nilai-nilai sebagai pengganti hirarki bidak

Nama	Nilai	Keterangan
Raja	6	Bidak paling penting yang harus dipertahankan
Ratu	5	Bidak yang lebih penting daripada bidak bernilai 4, 3, 2, dan 1 namun tidak lebih penting dari bidak bernilai 6
Benteng	4	Bidak lebih penting daripada bidak bernilai 3, 2, dan 1 namun tidak lebih penting dari bidak bernilai 6 dan 5
Pluncur	3	Bidak lebih penting daripada bidak bernilai 2 dan 1 namun tidak lebih penting dari bidak bernilai 6, 5, dan 4
Kuda	2	Bidak lebih penting daripada bidak bernilai 1 namun tidak lebih penting dari bidak bernilai 6, 5, 4, dan 3
Pion	1	Bidak bernilai 1 yang dianggap tidak lebih penting daripada bidak bernilai 6, 5, 4, 3, dan 2

Berdasarkan kriteria pada tingkatan hirarki tersebut, dapat ditentukan nilai bagi

Jurnal Algoritma, Logika dan Komputasi
Vol.I (No. 2) : 68 - 81. Th. 2018
p-ISSN: 2620-620X
e-ISSN: 2621-9840

masing-masing yang dapat mendefinisikan kepentingan para bidak yang dapat digunakan dalam evaluasi pada perangkat lunak.

Pada penelitian ini masing-masing bidak akan diberikan nilai 25 dengan alasan bahwa nilai tersebut sudah mendekati nilai yang didapat dari kemungkinan pergerakan terbanyak yang dapat dilakukan oleh bidak ratu yaitu 27, untuk membedakan hirarki antara bidak maka nilai 25 akan dikalikan dengan jumlah dari cara gerak yang dapat dilakukan oleh bidak tersebut.

Tabel 4. Nilai untuk masing-masing bidak kecuali bidak raja

Nama	Jumlah gerak * nilai awal	Hasil
Ratu	27 * 25	625
Benteng	15 * 25	375
Pluncur	13 * 25	325
Kuda	8 * 25	200
Pion	4 * 25	100

Dikarenakan raja memiliki nilai terbesar berdasarkan kepentingan raja dalam permainan, dan tidak mempedulikan sebanyak apapun raja dapat bergerak, raja tetap yang terpenting dalam permainan catur, untuk bidak raja diberikan nilai 100.000, nilai tersebut akan mengubah hasil evaluasi yang cukup signifikan jika raja tersebut terbunuh yang dapat menyebabkan berakhirnya permainan.

Dengan adanya penilaian bagi masing-masing bidak maka dapat diketahui untuk keadaan pertama atau keadaan pembukaan dalam permainan, di mana masing-masing pemain mendapatkan nilai dari evaluasi bidak dengan rincian berikut:

B = {b | jumlah pada bidak yang akan dihitung}
N = {n | nilai bagi bidak yang akan dihitung}
EB = {eb | n * b}

Jika G = raja, Q = ratu, R = benteng, K = kuda, B = pluncur dan P = pion, maka untuk masing-masing bidak yaitu:

$E_{(g)}B_{(g)}$ = 100.000 * 1 = 100.000
$E_{(q)}B_{(q)}$ = 625 * 1 = 625
$E_{(r)}B_{(r)}$ = 375 * 2 = 750
$E_{(b)}B_{(b)}$ = 325 * 2 = 650
$E_{(k)}B_{(k)}$ = 200 * 2 = 400
$E_{(p)}B_{(p)}$ = 100 * 8 = 800

Sehingga untuk menentukan nilai yang didapat oleh evaluasi bidak adalah sebagai berikut:

$$EB = E_{(g)}B_{(g)} + E_{(q)}B_{(q)} + E_{(r)}B_{(r)} + E_{(b)}B_{(b)} + E_{(k)}B_{(k)} + E_{(p)}B_{(p)}$$

$$EB = 100.000 + 625 + 750 + 650 + 400 + 800$$
$$EB = 103.225$$

Nilai 103.225 menjadi nilai bagi masing-masing pemain dari hasil perhitungan evaluasi bidak yang didapat pada saat pembukaan permainan dan dengan kondisi belum ada satupun bidak milik kedua pemain yang telah terbunuh, nilai tersebut akan relatif berubah selama permainan berlangsung seperti jika ada bidak yang terbunuh atau terdapat bidak pion yang telah mencapai baris paling depan yang dapat ditukar oleh bidak lain sesuai keinginan pemain.

Komputer akan menghitung jumlah dari evaluasi posisi dan evaluasi bidak, dan selanjutnya dalam proses pohon pencarian komputer akan membandingkan nilai hasil hitung evaluasi pada satu *node* dengan *node* lainnya. berikut ini adalah rumus untuk mengevaluasi setiap *node-node* dalam pohon pencarian:

$$K = EP_{(k)} + EB_{(k)}$$
$$M = EP_{(m)} + EB_{(m)}$$
$$N = K - M$$

Di mana K = komputer, M = manusia, EP = evaluasi posisi, EB = evaluasi bidak dan N = nilai evaluasi, sehingga didapatkan nilai N dari setiap *node* yang akan dibandingkan dengan nilai N pada *node* lainnya, pohon pencarian akan mencari nilai (N) terbesar untuk pemain komputer dan akan mencari nilai (N) terkecil untuk pemain pengguna.

Secara lengkap urutan mekanisme yang perlu dilakukan dalam mengimplementasikan prosedur pohon

Jurnal Algoritma, Logika dan Komputasi
Vol.I (No. 2) : 68 - 81. Th. 2018
p-ISSN: 2620-620X
e-ISSN: 2621-9840

pencarian menggunakan algortima *minimax* pada permainan catur adalah:

a. Asumsikan bahwa kedua pemain akan mencari nilai terbaik untuk dirinya masing-masing, di mana komputer akan mendapatkan nilai max dan pemain manusia akan mendapatkan nilai min.
b. Tentukan ke dalaman pencarian untuk *node* yang akan dievaluasi, definiskan 2 variabel semisal *i* dan *u*, dimana *i* = *u* = jumlah ke dalaman.
c. Cari dan tentukan jumlah kemungkinan gerak yang dapat dilakukan oleh bidak yang menghasilkan *subnodes* dari *node* yang sedang berlangsung, definisikan variabel *p,* dimana *p* = pergerakan, *n* = 0, dan *j* = jumlah pergerakan.
d. Lakukan pergerakan $p_{(i)(n)}$.
e. Jika *i>1* kurangi *i* dengan 1 sehingga *i* = *i-1* lalu lakukan tindakan mulai dari poin c.
f. Lakukan evaluasi nilai total, jika *n = 0* maka definisikan variabel semisal *x* = nilai evaluasi, jika tidak bandingkan $x_{(n)}$ dengan $x_{(n-1)}$, untuk pencarian *max* maka ambil nilai *x* yang lebih besar sebagai pengganti terhadap nilai *x*, untuk pencarian *min* maka ambil nilai *x* yang lebih kecil sebagai pengganti terhadap nilai *x*.
g. Jika *i=1* dan jika *n<j* maka tambahkan nilai *n* dengan 1 sehingga *n = n+1*, lalu lakukan lagi tindakan pada poin d.
h. Jika *n=j* dan *i = 1*, maka tambahkan nilai *i* dengan 1 sehingga *i=i+1* dan *n = 0* lalu berikan nilai *x* terhadap *x* pada *node* $p_{(i)}$
i. Lakukan lagi pergerakan pada poin d.
j. Jika *i=u*, maka pada simpul tersebut akan didapatkan nilai dari *x* yang terbaik yang merupakan hasil pencarian keseluruhan dari prosedur algoritma *minimax*, di mana *x* akan mengandung nilai *min* atau *max*.

Sedangkan dalam *alpha-beta pruning* prosedur-prosedur yang perlu dilakukan adalah:

a. Asumsikan bahwa kedua pemain sama-sama mencari nilai terbaik untuk dirinya masing-masing, di mana pemain manusia mencari nilai *min* dan pemain komputer mencari nilai *max*.
b. Tentukan ke dalaman pencarian untuk *node* yang akan dievaluasi, definiskan 4 variabel semisal *i, u*, α dan β dimana *i* = *u* = jumlah ke dalaman, α = *alpha* = ∞ dan β = *beta* = $-\infty$
c. Cari dan tentukan jumlah kemungkinan gerak yang dapat dilakukan oleh bidak yang menghasilkan *subnodes* dari *node* yang sedang berlangsung, definisikan variabel *p,* dimana *p* = pergerakan, *n* = 0, dan *j* = jumlah pergerakan.
d. Lakukan pergerakan $p_{(i)(n)}$.
e. Jika *i>1* kurangi *i* dengan 1 sehingga *i* = *i-1* lalu lakukan tindakan mulai dari poin c.
f. Lakukan evaluasi nilai total, jika *n = 0* maka definisikan variabel semisal *x* = nilai evaluasi, jika tidak lakukan pengecekan batas α atau β, untuk pencarian pada *node min*, jika α *!=* ∞ bandingkan nilai *x* dengan nilai α, jika *x*<α maka lakukan tindakan pada poin *h*, jika pencarian pada *node max* jika β *!=* $-\infty$ maka bandingkan nilai *x* dengan nilai, β jika *x*>β maka lakukan tindakan pada poin h, jika *x*>α atau *x*<β maka bandingkan $x_{(n)}$ dengan $x_{(n-1)}$, untuk pencarian *max* ambil nilai yang terbesar sebagai pengganti terhadap nilai *x*, untuk pencarian *min* ambil nilai terkecil sebagai pengganti terhadap nilai *x*.
g. Jika *i=1* dan jika *n<j* maka tambahkan nilai *n* dengan 1 sehingga *n = n+1*, lalu lakukan lagi tindakan pada poin d.
h. Jika *n=j* dan *i = 1*, maka tambahkan nilai *i* dengan 1 sehingga *i=i+1* dan *n = 0* lalu berikan nilai *x* terhadap *x* pada *node* $p_{(i)}$
i. Jika pencarian yang berlangsung berada pada *node max* maka berikan nilai terhadap β = *x*, jika pencarian pada *node min* maka berikan nilai terhadap α = *x*.
j. Lakukan lagi pergerakan pada poin d.
k. Jika *i=u*, maka pada simpul tersebut akan didapatkan nilai dari *x* yang terbaik yang merupakan hasil pencarian keseluruhan dari prosedur algoritma *minimax*, di mana *x* akan mengandung nilai *min* atau *max*.

Jurnal Algoritma, Logika dan Komputasi
Vol.I (No. 2) : 68 - 81. Th. 2018
p-ISSN: 2620-620X
e-ISSN: 2621-9840

Urutan prosedur *alpha-beta pruning* tersebut memiliki kemiripan dengan algoritma *minimax*, namun bagian yang berbeda dan sekaligus menjadi penting dalam prosedur *alpha-beta pruning* adalah saat nilai batas α dan β diinisialisasi yang kemudian nilai dari batasan dibandingkan dengan hasil evaluasi pada *node-node* lainnya, sehingga ketika nilai evaluasi melewati batasan maka pencarian evaluasi akan dipangkas dan dilanjutkan dengan evaluasi *node* di atasnya.

Dalam ilmu komputer teknik pengurutan memiliki banyak pilihan, salah satunya dengan menggunakan algoritma *bubble sort,* algoritma ini merupakan metode pengurutan yang paling sederhana di antara metode-metode pengurutan lainnya [20, p. 164]. Dalam algoritma *bubble sort* setiap bilangan yang dihasilkan akan dibandingkan mulai dari urutan pertama dengan bilangan pada urutan setelahnya, dalam pengurutan berjenis *descending* (mulai dari yang terbesar hingga terkecil), jika bilangan setelahnya lebih besar dari bilangan sebelumnya maka nilai tersebut akan ditukar posisinya dengan bilangan sebelumnya, lalu perulangan fungsi perbandingan akan dilakukan terus hingga setiap data pada koleksi dibandingkan, sehingga data yang sebelumnya tidak terurut menjadi terurut. Sedangkan dalam pengurutan berjenis *ascending* (dari mulai yang terkecil hingga terbesar) lawan dari *descending* jika nilai pada bilangan setelahnya lebih kecil dari bilangan sebelumnya maka nilai tersebut akan ditukar posisinya dengan nilai sebelumnya, pengurutan akan dilakukan terhadap node atas pada kedalaman pencarian level 1 dan tidak diterapkan terhadap *subnodes* di bawahnya atau pada level yang lebih besar dari level 1

Representasi pemrograman C++ untuk algoritma *bubble sort* dinyatakan sebagai berikut [21, p. 7]:

```
for (int i = 0; i < numbers.length-1; i++){
    int min = i;
        for (int j = i+1; j < numbers.length; j++){
            if (numbers[j] < numbers[min]){
                min = j;
            }
    }
    int temp = numbers[min];
    numbers[min] = numbers[i];
    numbers[i] = temp;
}
```

Sedangkan untuk mekanisme metode *transposition table* dan *history heuristic* diimplementasikan dengan memanfaatkan teknik penyimpanan yang telah menjadi objek *hashmap* dalam *library* dari pemrograman java [22, pp. 113-115]. *Hashing* adalah algoritma yang dibuat bertujuan agar pencarian data lebih cepat, *hashing* juga merupakan perbaikan terhadap algoritma *binary search* dan *interpolation search* [20, p. 238]. Pembandingan data dalam *hash* dapat diatasi oleh perhitungan matematis, salah satunya dengan membiarkan beberapa nilai keadaan dari data ke dalam kunci *hash* di mana komputer dapat dengan akurat membandingkan beberapa data hanya dengan menggunakan kunci *hash* dan bukan membandingkan dengan isi data, salah satu cara untuk membentuk kunci *hash* adalah *digit selection*, untuk menentukan nilai apa yang akan digunakan menjadi kunci dalam *digit selection*, maka perlu diketahui karakteristik dari data yang akan disimpan.

Berikut ini adalah rincian elemen-elemen dari karakterisitik data dalam permainan catur yang akan dijadikan sebagai komposisi kunci ke dalam *hashmap, yaitu*:

- Lokasi kolom, contoh nilai untuk kolom D:2 adalah 42
- Jenis bidak, contoh nilai untuk bidak kuda adalah 2
- Pemilik bidak, untuk bidak komputer mendapat nilai 1, dan untuk bidak manusia mendapat nilai 0.
- Langkah giliran, jika giliran komputer maka bernilai 1 jika langkah giliran pengguna maka bernilai 0.
- Hirarki bidak, nilai dari hirarki untuk masing-masing bidak
- Level, contoh level nilai 2.

Jurnal Algoritma, Logika dan Komputasi
Vol.I (No. 2) : 68 - 81. Th. 2018
p-ISSN: 2620-620X
e-ISSN: 2621-9840

Komposisi tersebut berdasarkan data bagi masing-masing bidak sehingga ketika sebuah data mewakili semua posisi bidak yang akan disimpan ke dalam tabel maka semua nilai bagi masing-masing bidak akan dikalkulasikan untuk mendapatkan nilai kunci *hash*, rumus untuk mendapatkan nilai kunci *hash* adalah:

N = Jumlah kolom dan baris
K = Lokasi kolom dan baris
H = Hirarki atau jenis bidak
C = {c | Bidak milik komputer = 1 atau pengguna = 0}
G = {g | langkah giliran komputer = 1 atau pengguna = 0}
L = Level
E =

$$\sqrt{(K_n * 1000) + (H * 100) + (C * 10) + (G * 1)}$$

HK =

$$E_0 * 1 + E_1 * 2 + E_2 * 3 + \ldots, \ldots, + E_{n-1} * N$$

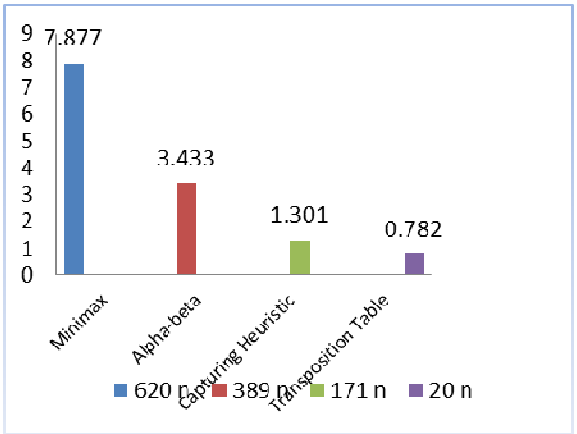
Di mana N = jumlah kolom, *K* = lokasi kolom, *H* = jenis bidak, *C* = pemilik bidak, *G* = giliran melangkah, dan *L* = level, dengan kondisi jika *C* adalah pemain komputer maka *C* = *1* dan jika C adalah pemain manusia maka *C* = *0*.

HASIL DAN PEMBAHASAN

Pengimplementasian dalam penelitian ini diterapkan ke dalam media android device dengan spesifikasi *minimum operating system* 2.3 *Gingerbread*, spesifikasi dari *processing unit* yang digunakan untuk pengujian yaitu CPU Dual-core 1.3 GHz Cortex-A7, *memory* 512 MB RAM. Bahasa pemrograman yang digunakan untuk *android device* dalam penelitian ini yaitu pemrograman java.

Percobaan perbandingan pertama dilakukan terhadap permainan catur antara pengguna melawan komputer, pergerakan dalam percobaan ini adalah pergerakan pembukaan permainan di mana semua bidak masih berada pada posisi awal di atas papan catur, pengguna akan menggerakkan bidak pion dari petak E2 menuju petak E4 dengan kedalaman level pencarian 2.

Pada gambar 5 dapat terlihat bahwa algoritma *minimax* menghasilkan jumlah pencarian sebanyak 620 node dengan kecepatan proses pencarian selama 7877 ms, sedangkan *alpha-beta pruning* menghasilkan jumlah pencarian sebanyak 389 node dengan kecepatan proses selama 3433 ms, *capturing heuristic* menghasilkan jumlah pencarian sebanyak 171 node dengan kecepatan proses pencarian selama 1301 ms, dan yang terakhir dalam percobaan kali ini adalah metode *transposition table* dengan jumlah pencarian sebanyak 20 node dengan lama proses pencarian 0.782 ms. Percobaan kali ini tidak menyertakan metode *history heuristic* dikarenakan tidak menghasilkan perubahan apapun setelah perubahan proses dari *transposition table,* untuk mengetahui hasil dari metode *history heuristic* percobaan dilakukan dengan mengganti level menjadi level 3, hal ini diketahui karena metode *history heuristic* mengambil data yang dapat diketahui dan diakibatkan oleh pencarian lebih besar dari kedalaman level 2 dari pencarian sebelumnya.



Gambar 5. Grafik hasil perbandingan metode untuk percobaan pertama

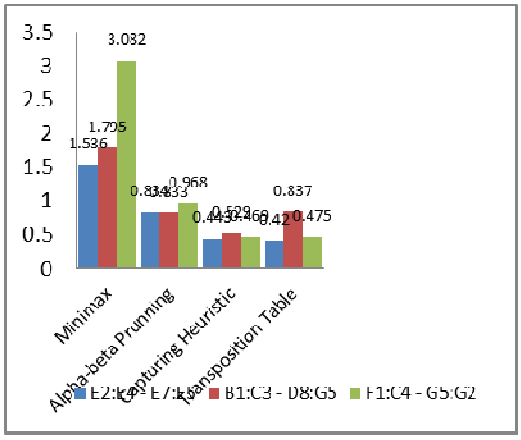
Tabel 5. Percobaan dengan history heuristic

No	P	K	Pencarian	
			N	D
1	G2:G3	D8:E7	0	00:00:344
2	D1:F3	D8:G5	4604	00:090:261

Jurnal Algoritma, Logika dan Komputasi
Vol.I (No. 2) : 68 - 81. Th. 2018
p-ISSN: 2620-620X
e-ISSN: 2621-9840

Pembuktian metode dari *history heuristic* hanya dapat diketahui setelah kedua pemain menggerakkan bidaknya, hal ini disebabkan pada pergerakan pertama atau pergerakan pembukaan di dalam *hashmap* belum tersimpan data pencarian, sehingga metode *capturing heuristic* tidak efektif pada pergerakan pertama. Pada tabel 5 baris pertama menunjukkan bahwa pohon pencarian tidak mengevaluasi satupun node dalam pohon pencarian, hal itu merupakan hasil optimasi dari implementasi *history heuristic*.

Percobaan kedua dilakukan dengan melakukan 3 pergerakan bidak milik pengguna yaitu pergerakan pion E2:E4, kuda B1:C3, dan pluncur F1:C4 dengan kedalaman level 2, pada gambar 6 menunjukkan grafik dari hasil percobaan tersebut.



Gambar 6. Grafik hasil perbandingan metode untuk percobaan kedua

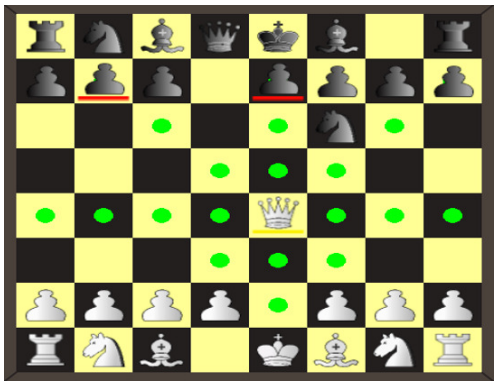
Tabel 6. Rincian hasil dari percobaan kedua

No	M	P	K	Pencarian	
				N	D
1	MX	E2:E4	E7:E5	620	00:01:536
2		B1:C3	D8:G5	926	00:01:795
3		F1:C4	G5:G2	1440	00:03:082
1	MX + AB	E2:E4	E7:E5	389	00:00:834
2		B1:C3	D8:G5	241	00:00:833
3		F1:C4	G5:G2	409	00:00:968
1	MX + AB + CH	E2:E4	E7:E5	171	00:00:443
2		B1:C3	D8:G5	224	00:00:529
3		F1:C4	G5:G2	154	00:00:469
1	MX + AB + CH + TT	E2:E4	E7:E5	20	00:00:420
2		B1:C3	D8:G5	224	00:00:837
3		F1:C4	G5:G2	154	00:00:475

M = Metode, P = Pengguna, K = Komputer, N = Node, D = Durasi, MX = *Minimax*, AB = *Alpha-beta*, CH = *Capturing Heuristic*, TT = *Transposition Table*

Pada Gambar 6 menunjukkan bahwa hasil dari optimasi setiap metode memberikan kecepatan proses pencarian yang lebih baik daripada hasil dari proses dengan hanya menggunakan metode algoritma *minimax* saja, sebagai rincian hasil proses dari percoban tersebut ditunjukkan pada Tabel 6.

Percobaan dilakukan kembali dengan kedalaman pencarian level 3 untuk mengetahui efektifitas dari metode *capturing heuristic* terhadap pohon pencarian, pergerakan kali yang dilakukan oleh pengguna kali ini adalah pergerakan yang bermula dari keadaan papan catur yang berbeda dari percobaan sebelumnya, pada Gambar 7 menunjukkan keadaan pada papan catur, di mana giliran pelangkah saat ini adalah bidak putih milik pengguna dan bidak ratu putih sedang terancam oleh kuda hitam milik pemain komputer.



Gambar 7. Keadaaan awal pada percobaan ketiga

Jurnal Algoritma, Logika dan Komputasi
Vol.I (No. 2) : 68 - 81. Th. 2018
p-ISSN: 2620-620X
e-ISSN: 2621-9840

Pada Gambar 7 menunjukkan keadaan di mana ratu bidak putih terancam, giliran pergerakan kali ini adalah milik bidak putih, pengguna sebagai pemilik bidak putih akan menggerakkan ratu dengan pergerakan E4:A4 yang bertujuan mengancam raja bidak hitam, dan rincian pergerakan selanjutnya ditunjukkan pada Tabel 7.

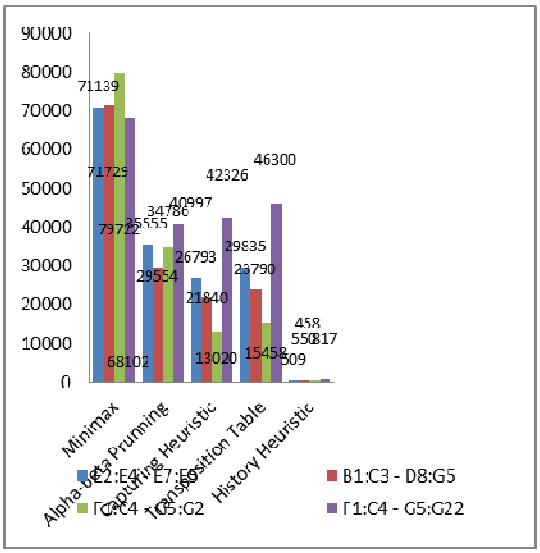
Tabel 7. Rincian hasil dari percobaan ketiga

No	M	P	K	Pencarian	
				N	D
1	MX	E4:A4	B8:C6	47961	01:11:139
2		F1:B5	C8:G4	47065	01:11:729
3		B5:C6	B7:C6	52734	01:19:722
4		A4:C6	G4:D7	47213	01:08:102
1	MX + AB	E4:A4	B8:C6	22755	00:35:555
2		F1:B5	C8:G4	18597	00:29:554
3		B5:C6	B7:C6	22220	00:34:786
4		A4:C6	G4:D7	27344	00:40:977
1	MX + AB + CH	E4:A4	B8:C6	16902	00:26:793
2		F1:B5	C8:G4	12832	00:21:840
3		B5:C6	B7:C6	8104	00:13:020
4		A4:C6	G4:D7	27612	00:42:326
1	MX + AB + CH + TT	E4:A4	B8:C6	16902	00:29:835
2		F1:B5	C8:G4	12832	00:23:790
3		B5:C6	B7:C6	8104	00:15:458
4		A4:C6	G4:D7	27612	00:46:300
1	MX + AB + CH + TT + HH	E4:A4	B8:C6	34	00:00:509
2		F1:B5	C8:G4	34	00:00:550
3		B5:C6	B7:C6	34	00:00:458
4		A4:C6	G4:D7	34	00:00:817

M = Metode, P = Pengguna, K = Komputer, N = Node, D = Durasi, MX = *Minimax*, AB = *Alpha-beta*, CH = *Capturing Heuristic*, TT = *Transposition Table, HH = History Heuristic*

Pada Gambar 8 menunjukkan grafik hasil dari percobaan ketiga seperti yang ditunjukkan pada Tabel 6 dengan kedalaman pencarian level 3.

Dapat dilihat pada gambar 8 dan tabel 7 bahwa metode *alpha-beta pruning* dapat memperkecil jumlah pencarian sehingga mengakibatkan proses pencarian menjadi lebih cepat, metode *capturing heuristic* juga mampu memangkas jumlah pencarian dalam percobaan tersebut, sedangkan pada metode *transposition tabel* tidak terdapat perbedaan jumlah node dan pada metode history heuristic, pemangkasan tidak terjadi akibat dari optimasi metode tersebut melainkan hasil dari optimasi *transposition table,* hal tersebut dapat diketahui karena jumlah node yang terdapat pada percobaan tersebut berjumlah 34 sedangkan seharusnya pemangkasan history heuristic dapat menyebabkan jumlah pencarian berjumlah 0.



Gambar 8. Grafik hasil perbandingan dari percobaan ketiga

SIMPULAN

Berdasarkan penelitian dan implementasi yang dilakukan maka dapat diambil kesimpulan bahwa metode *alpha-beta pruning*, *capturing heuristic*, *transposition table dan history heuristic* dapat meminimalkan jumlah simpul pencarian sehingga dapat mengoptimasi proses dari

Jurnal Algoritma, Logika dan Komputasi
Vol.I (No. 2) : 68 - 81. Th. 2018
p-ISSN: 2620-620X
e-ISSN: 2621-9840

algoritma *minimax*. Sebagai perbandingan, metode *transposition table* menjadi metode terbaik yang dapat memangkas pencarian dari algoritma minimax dengan jumlah perbedaan yang cukup signifikan, namun optimasi yang dapat dilakukan oleh *transpositon table* dan juga *history heuristic* hanya dapat terjadi jika data pergerakan yang hendak dievaluasi telah tersimpan dalam hashmap.

DAFTAR PUSTAKA

[1] A. M. Gatej, "Chess Cryptosystem," Journal of Mobile, Embedded and Distributed Systems, pp. 29-34, 2010.

[2] F. Gobet and G. Campitelli, "Educational Benefits of Chess Instruction: A Critical Review," in In T. Redman (Ed.), Chess and Education: Selected essays from the Koltanowski conference, Dallas, 2006.

[3] F. Gliga and P. I. Flesner, "Cognitive Benefits of Chess Training in Novice Children," Procedia - Social and Behavioral Sciences, vol. 116, pp. 962-967, 2014.

[4] W. M. Bart, "On the Effect of Chess Training on Scholastic Achievement," Frontiers in Psychology, vol. 5, no. 762, pp. 1-3, 2014.

[5] M. Scholz, H. Niesch, O. Steffen, B. Ernst, M. Loeffler, E. Witruk and H. Schwarz, "Impact of Chess Training on Mathematics Performance and Concentration Ability of Children with Learning Disabilities," International Journal of Special Education, vol. 23, no. 3, pp. 138-148, 2008.

[6] Z. Halim, A. R. Baig and K. Zafar, "Evolutionary Search in the Space of Rules for Creation of New Two-Player Board Games," International Journal on Artificial Intelligence Tools, vol. 23, no. 2, pp. 1-26, 2014.

[7] Suyanto, Artificial Intelligence, Bandung: Informatika Bandung, 2014.

[8] O. E. David, H. J. v. d. Herik, M. Koppel and N. S. Netanyahu, "Genetic Algorithms for Evolving Computer Chess Programs," IEEE Transactions on Evolutionary Computation, vol. 18, no. 5, pp. 779 - 789, 2014.

[9] C. E. Shannon, "Programming a Computer for Playing Chess," Philosopical Magazine, vol. 41, pp. 256-275, March 1950.

[10] A. Weinstein, M. Littman and S. Goschin, "Rollout-based Game-tree Search Outprunes Traditional Alpha-beta," in 10th European Workshop on Reinforcement Learning, Edinburgh, 2012.

[11] D. E. Knuth and R. W. More, "An Analysis of Alpha-Beta Prunning," Artificial Intelligence, vol. 6, pp. 293-326, 1975.

[12] Y. Liu, H. Zang and P. Fu, "A Hybrid Game Tree Search Algorithm for Playing Chess," Journal Of Computational Information Systems, vol. 14, p. 5803–5811, 2012.

[13] R. G. Ulma, M. Fusco and A. Mycroft, Java 8 in Action, Shelter Island: Manning Publication, 2014.

[14] T. Sutojo, E. Mulyanto and V. Suhartono, Kecerdasan Buatan, Yogyakarta: Andi Yogyakarta, 2011.

[15] S. J. Russel and P. Norvig, Artificial Intelligence a Modern Approach, New Jersey: Prentice Hall, 1995.

[16] G. Weis, A Modern Approach to Distributed Artificial Intelligence, London: The MIT Press, 1999.

[17] B. Robertie, I. Horowitz and H. Kidders, Cara Cepat dan Mahir Bermain Catur, 1 ed., Semarang: Dahara Prize, 2014.

[18] R. Bott and M. Stanley, Discovering Chess, London: Faber And Faber, 1975.

[19] J. R. Capablanca, Chess Fundamentals, New York: Harcourt, Brace & World Inc, 1934.

[20] E. B. Purwanto, Perancangan dan Analisis Algoritma, Yogyakarta: Graha Ilmu, 2008.

[21] A. Taherkhani, L. Malmi and A. Korhonen, "Using Roles of Variables in Algorithm Recognition," in International Conference on Computing Education Research, Uppsala, 2009.

Jurnal Algoritma, Logika dan Komputasi
Vol.I (No. 2) : 68 - 81. Th. 2018
p-ISSN: 2620-620X
e-ISSN: 2621-9840

[22] B. Carlstrom, J. W. Chung, H. Chafi, A. McDonald, C. C. Minh, L. H. C. Kozyrakis and K. Olukotin, "Executing Java Program with Transactional Memory," Science of Computer Programming, pp. 111-129, 2005.